Departamento de Informática

PROYECTO FIN DE CARRERA

Desarrollo de la lógica de un videojuego de plataformas en Android

Autor: Juan Luis Campíns Frau

Tutor: Raúl Arrabales Moreno

Leganés, Enero de 2012

Agradecimientos

Quería agradecer a Jorge Muñoz Fuentes por los buenos y experimentados consejos proporcionados, y su dedicación.

Al "*Equipo*", porque cada práctica hecha juntos, cada hora en la biblioteca estudiando *a tope*, cada momento que me proporcionaban su ayuda sin pedirlo, cada examen sintiendo su apoyo, han hecho que con su fuerza y amistad, me haya ayudado a llegar hasta aquí.

A Rocío, por aparecer en mi vida.

A mi familia, los más importantes, por su confianza, apoyo incondicional y ánimos…siempre estáis ahí.

Y por último, quería recordar con especial cariño a mi abuela, puesto que este proyecto está dedicado a ella. Fuiste ejemplo de lucha y fuerza, "*¡Ja he acabat!*".

Resumen

En la actualidad, la industria del videojuego es uno de los sectores más rentables y estables a nivel mundial, cuyo crecimiento sigue en aumento conforme transcurren los años. Uno de los factores que han permitido al sector coger aún más impulso si cabe, ha sido la expansión hacia otras plataformas como son los smartphones, debido al crecimiento de su popularidad entre los usuarios**.**

Este proyecto tiene como objetivo el desarrollo de la lógica de un videojuego del género plataformas en smartphones que dispongan del sistema operativo *Android*. Todo esto abarca el manejo del personaje principal, de los enemigos, escenarios, colisiones entre los elementos, el ciclo de vida, etc. Como puntos principales del videojuego se destacan el uso de una biblioteca desarrollada por otro PFC como interfaz gráfica de la lógica del proyecto. Ésta permite que el entorno donde se desarrolla el videojuego sea en 3D, por lo que el personaje que controla el usuario puede avanzar en profundidad, además de su desplazamiento horizontal. Y el empleo de uno de los sensores del dispositivo móvil. Su uso permite conseguir que con un simple giro del terminal se aumente la perspectiva de visualización del escenario, o que gire en el mismo sentido en el que el usuario lo haga con el terminal.

Palabras clave: smartphone, Android, videojuego, plataformas, 3D, lógica

Abstract

Nowadays, the videogame industry is one of the most profitable and stable sectors at world-wide level, whose growth increases as the years pass. One of the factors that have allowed the sector to take impulse, has been the expansion towards other platforms as they are smartphones, due to the growth of their popularity among users.

This project has like objective the development of the logic of a platforms videogame genre in smartphones that has the *Android OS*. This covers the handling of the main character, of the enemies, scenarios, collisions between the elements, the life cycle, etc. As main points of videogame, stand out the use of a library developed by another PFC like GUI of the logic of the project. This one allows that the environment where the videogame is developed is in 3D, so the character that the user controls can advance in depth, in addition to his horizontal displacement. And the use of one of the sensors of the mobile device. Its use allows the increase of the viewing perspective of the scenarios with a simple turn of the terminal, or it turns in the same direction in which the user does with the terminal.

Keywords: smartphone, Android, video game, plattforms, 3D, logic

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Capítulo 1

Introducción y objetivos

1.1 Introducción

Sin duda alguna, el siglo XX fue el punto de partida de una gran evolución informática, del que hasta el día de hoy, se producen continuamente importantes avances.

El gran invento que inició esta revolución fue el computador u ordenador, que a mediados de ese siglo se presentaba como una enorme máquina capaz de procesar datos y que solo se encontraba disponible para el ejército o importantes universidades, en el que la investigación era su principal cometido. Según el cofundador de *Intel*[1], *Gordon Moore*[2][3]:

> "*el número de transistores de un chip se duplica cada dos años*"

…por lo que estas grandes máquinas evolucionaron ágilmente a tamaños más manejables, menor peso, mayor capacidad y procesamiento, precios más asequibles, con la consecuente rápida expansión entre las masas.

Años después se incorporó a los ordenadores *Internet*, que al igual que éstos, primero fue de uso privado por parte de universidades, investigadores, científicos, profesores y también militar, para posteriormente introducirse en el ámbito público consiguiendo ofrecer un servicio abierto a principios de la década de 1990. Esta tecnología ha pasado a formar parte de cada uno de nosotros tanto en nuestra vida diaria, laboral y académica.

Siguiendo en la misma década de los 90, la red móvil había evolucionado mucho desde sus comienzos en la *II Guerra Mundial*[4], y se encontraba en una fase de

perfeccionamiento, de búsqueda de novedades y mejoras, debido a la importancia que supone la comunicación a distancia. Estas redes se han ido desarrollando conforme pasan los años, ofreciendo más servicios, mayor velocidad de comunicación y de transmisión. Son utilizadas por los aparatos denominados móviles. En sus inicios eran grandes, algo pesados, solo podían realizar llamadas y enviar mensajes de texto, pero han ido incorporando paulatinamente más servicios, recursos y posibilidades. Consecuentemente, esos primeros móviles en el que su cometido primordial era el de la comunicación, pasaría a un segundo plano tras las incorporación de todas las evoluciones, por lo que comenzarían a denominarse dispositivos móviles.

Los smartphones[5] son a día de hoy, los dispositivos móviles más evolucionados, debido a la reunión bajo su reducido tamaño de todos esos avances. Pueden usar las redes móviles para conectarse a *Internet*, tienen capacidad de procesamiento, de gestión de memoria, de control de hardware, de almacenamiento, *GPS*[6], pantallas táctiles, distintos sensores, aplicaciones de usuario que solo se encontraban en los ordenadores, como correo, aplicaciones ofimáticas o videojuegos… todo bajo el control de un Sistema Operativo[7], adaptado específicamente a sus características, como un pequeño ordenador. Todo esto supone que el smartphone haya conseguido ser el primer dispositivo en desbancar a los ordenadores en ventas en el último trimestre de 2010[8].

Centrándonos en los distintos Sistemas Operativos que controlan los smartphones, existen actualmente gran variedad de ellos, pero tres son los favoritos entre los usuarios a la hora de escoger un terminal móvil. Estos son: *iOS*[9], que posee una cuota de mercado a escala mundial de smartphones aproximada del 17%, por delante se encuentra *Symbian OS*[10] con un cuota aproximada del 27% y finalmente *Android*[11], que ostenta aproximadamente el 36%, datos del primer trimestre de 2011[12].

Este último, *Android*, en el año 2010 tenía nada más que el 10% de esa cuota de mercado y era cuarto en el ranking, por lo que su crecimiento es notorio. Se caracteriza por ser el principal producto de la *Open Hanset Alliance*[13], un consorcio de compañías, la cual fue comprado por *Google*[14] en julio de 2005, que bajo la licencia *Apache*[15], liberó la mayoría del código de *Android* y pasó a ser de licencia libre. Está basado en *Linux*[16] y todas sus aplicaciones se escriben en lenguaje *Java*[17], disponiendo además de una máquina virtual específica llamada *Dalvik*[18], gran cantidad de bibliotecas escritas en C además de una *API* gráfica llamada *OpenGL ES*[19] para el desarrollo de entornos de 2D como para de 3D. Representa toda una pila de software para dispositivos móviles, aunque ahora también se desarrolla para tablets, PCs etc.

Es aquí donde el presente proyecto se establece, el cual trata sobre el desarrollo de un videojuego usando el smartphone como dispositivo, que unifica todas las evoluciones informáticas que han surgido durante estos años; *Android* como sistema operativo en su versión igual o superior a la 2.2 o su correspondencia en la versión de SDK[20] a la 8; y finalmente *OpenGL ES* como biblioteca gráfica en su especificación 2.0 de la API[21], debido a que es compatible con su versión anterior 1.1 y con la mayoría de los dispositivos *Android*.

Es un videojuego de plataformas, en el que el personaje que controla el usuario tiene que completar la totalidad de los niveles que se le presenten, habiendo recogido con anterioridad la llave que le abre la puerta al siguiente escenario. Mediante el uso de los

controles que se muestran en pantalla podrá progresar, retroceder, saltar y avanzar en profundidad debido a que los escenarios donde se desarrolla el videojuego es en 3D. Tiene que esquivar todo aquello que le impida llegar a ese fin, pudiendo ser que el escenario se lo imposibilite con algún obstáculo, como elementos puntiagudos sobre los que no se debe dejar caer, meteoritos que caen de la parte superior del escenario, o que hacia donde se desea avanzar no exista plataforma alguna sobre la que apoyarse y caiga al vacío. También debe esquivar a los enemigos dispuestos sobre el escenario, que le perseguirán para eliminarle y de esta forma no conseguir llegar al fin de la pantalla, incluso pudiendo alguno de ellos solventar obstáculos.

Como se ha comentado con anterioridad, los escenarios no solo tienen componentes en horizontal y vertical, si no que al ser tridimensionales también tienen en profundidad. Debido a la gran cantidad de sensores que poseen los smartphones, haciendo empleo de uno de ellos se consigue mediante un simple giro del dispositivo móvil, poder visualizar más perspectiva de lo que a simple vista ofrece el escenario y ver los componentes dispuestos en profundidad. Por lo tanto, el usuario puede saber si existe plataforma donde apoyar al personaje hacia la dirección que desea desplazarlo.

Todo éste entorno 3D, al igual que su distribución, los componentes tridimensionales y el dibujado de éstos, es fruto del proyecto de fin de carrera desarrollado por una compañera, y que este videojuego lo utiliza como biblioteca. Por lo tanto, el cometido del presente proyecto es el del desarrollo de toda la lógica que conlleva un videojuego y los datos resultantes de todo ello son dibujados por pantalla gracias a la interfaz 3D desarrollada citada anteriormente.

1.2 Objetivos

El objetivo principal del presente proyecto es el de la creación de un videojuego de plataformas destinado a dispositivos móviles con S. O. *Android*. Este usará la biblioteca que ha creado una compañera como PFC, que emplea *OpenGL ES* para la creación de un entorno tridimensional, y que servirá como interfaz de toda la lógica desarrollada en este proyecto. Mediante el aprendizaje del uso de uno de los sensores del dispositivo móvil y su implementación, se conseguirá que con un simple giro del terminal por parte del usuario, aumente la perspectiva de visualización que se tiene del escenario en el que se encuentre, o gire a éste en el mismo sentido en el que el usuario lo haga con el terminal. Asimismo, con la formación en los distintos eventos que se producen sobre la pantalla táctil del dispositivo móvil, se empleará para realizar determinadas acciones sobre el personaje.

Por encima de todo, el objetivo principal de la consecución de la lógica del videojuego planteado, implica el manejo de cantidad de datos, por lo que se dedicará mucho tiempo a pruebas y repeticiones de éstas para asegurarse que no conduce a error alguno y que cada pequeña unidad que lo conforma funciona exactamente de la manera que el usuario lo espera. Esto es para que el videojuego cumpla con sus expectativas y produzca la sensación de que funciona con normalidad, como otro juego existente en el mercado, dado que si se cae en la precipitación, puede que no encuentren interesante el videojuego y todo el tiempo y trabajo dedicado se considerará como perdido. Por lo

tanto, en la consecución del objetivo principal se debe obtener un resultado de la mayor calidad que se pueda.

Para la consecución del objetivo principal, primero se tiene que realizar el logro de unos objetivos secundarios, que son:

- Aprendizaje de todos los recursos, herramientas y posibilidades que ofrece el código de programación *Android*, como el empleo de los sensores, de los distintos eventos que se pueden capturar como los del teclado físico o pulsaciones sobre la pantalla. Por lo tanto, para llevar a cabo el desarrollo del videojuego sabré de qué manera emplearlos, qué utilizar en cada determinado momento y cómo conseguir un uso eficiente de los medios disponibles, conduciéndome a un código resultante estable y firme.

- El propio desarrollo de un videojuego en sí, debido a que en la carrera que he estudiado (*Ingeniería Técnica de Telecomunicaciones: Imagen y Sonido*) no había realizado implementación alguna anterior. Si que había aprendido algunas de las herramientas y lenguajes de programación necesarios para el desarrollo de un videojuego, por lo que a la hora de disponerme a ello, no sería lento ni costoso. Era una parte del conocimiento técnico inexplorada por mi, la cual no implica que no tuviera ganas en ello, al contrario, siempre gusta poder aprender más y sobre todo centrado en un mercado como el de los videojuegos en dispositivos móviles tan en crecimiento, y más si cabe sobre un sistema operativo en auge como es *Android*.

- El videojuego resultante debe proporcionar a terceros la posibilidad de realizar la extensión de éste, por lo que cualquiera que le guste y desee su ampliación y/o mejora, pueda a partir de la lectura de la documentación su realización. Consecuentemente deber estar redactada y descrita de tal manera que facilite este cometido.

- El presente proyecto se dispondrá para el uso de terceros debido a que *Android* dispone de una tienda de aplicaciones en línea administrada por *Google* con el nombre de *Android Market*[22], en el que la gran comunidad de desarrolladores que existe divulga sus aplicaciones para el uso de cualquiera.

- La realización del proyecto proporciona a su vez, la posibilidad de conocer y utilizar varias herramientas tales como *XML*[23], *OpenGL ES*, *Eclipse*[24], uso de dispositivos móviles virtuales…

1.3 Estructura del documento

El contenido de este documento sigue una estructura dividida en capítulos, explicando determinados apartados pertenecientes a este proyecto, los cuales se describen a continuación:

- **Introducción:** en él se da una primera idea de en qué consiste el proyecto, explicando las motivaciones que se tuvieron para escoger éste y los objetivos que se pretenden conseguir con él.

- **Estado del Arte:** se realiza un análisis de la historia del videojuego desde sus comienzos hasta el día de hoy, un estudio de la situación actual en el que se encuentran los videojuegos en *Android*, los distintos géneros de videojuegos que existen incluyendo un análisis de los juegos más exitosos y el por qué de su triunfo, qué herramientas se emplean para el desarrollo de éstos tanto las actuales como las que se podrán emplear en un tiempo próximo, y el futuro de los videojuegos en *Android*.

- **Descripción del videojuego:** como bien indica el nombre del capítulo, se efectúa una descripción del proyecto profundizando algo más en distintos aspectos que le caracteriza.

- **Desarrollo de la lógica de *Cube´s War*:** en este capitulo se analiza los requisitos de usuario y de software, se proporciona una explicación del diseño conceptual para tener una perspectiva global de los módulos que componen el videojuego, es decir, la arquitectura que la forma; toda la implementación del videojuego explicando entre otros: el diseño del jugador, sus movimientos, la captura de los eventos sobre la pantalla, la creación de los enemigos, la detección de las colisiones entre los distintos elementos del videojuego, diseño de los escenarios y sus elementos, menús, componentes gráficos, animaciones, sonidos, vibración, movimiento de la cámara…

- **Conclusiones:** se expone las distintas reflexiones que obtengo tras haber realizado este proyecto, y si se han conseguido los distintos objetivos que se habían planteado en un principio con él.

- **Trabajos futuros:** se describen las distintas ideas que pueden surgir para al ampliación y/o mejora de este videojuego final, de cómo incluir una mayor funcionalidad a éste y para que, en videojuegos ya implantados en el mercado o en nuevos, puedan reutilizarlo a fin de obtener una innovación en ellos.

- **Bibliografía y referencias:** se incluyen todas las distintas referencias y fuentes que se han consultado para la realización de este proyecto y memoria, además de los índices de figuras, tablas y códigos presentes en él.

- **Apéndices:** este capitulo abarca distintos apartados como son el de la planificación inicial del proyecto con toda la fase de documentación previa al desarrollo del videojuego; seguida de la planificación final del tiempo que se ha tardado en la realización de éste. También se incluye el desglose del presupuesto que ha conllevado el proyecto, de cómo y cuando recuperaría la inversión inicial asumida y finalmente el manual de usuario, en el que se explica los distintos pasos que hay que seguir para poder instalar la aplicación en un dispositivo móvil y probarla.

Capítulo 2

Estado del arte

La aplicación que se desarrolla en este proyecto es la de un videojuego de plataformas sobre el sistema operativo *Android* de dispositivos móviles. En este capítulo de la memoria, se tratará de dar una visión global de los videojuegos, desde sus comienzos en un osciloscopio hasta el día de hoy, pasando por las distintas plataformas en los que se han desarrollado, de esta manera se proporciona al lector la información necesaria para entender las peculiaridades del proyecto

Posteriormente, en el que ya se ha obtenido una idea general de los videojuegos, se analizarán éstos dentro del sistema operativo *Android*: cómo se pueden desarrollar, que tipos de juegos hay, cuál será su evolución en el futuro, el éxito que tienen…

2.1 Evolución de los videojuegos

Muchos consideran éste como el primer juego de la historia, pero no cumplía los requisitos para considerarse como tal, debido a que solo estaba formado por imágenes fijas por lo que no había movimiento alguno en la pantalla. Era el *OXO*[25], que fue la tesis doctoral de *Alexander Sandy Douglas* de la *Universidad de Cambridge*[26]. Es una versión del “*Tres en Raya*” o del "*Tic-Tac-Toe*”.

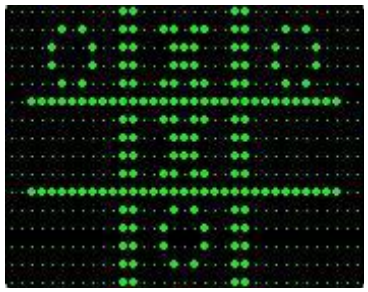


Figura 1. *OXO*

Se trataba de completar una fila de tres círculos o cruces en diagonal, horizontal o vertical, en una pantalla de 35x16 píxeles. No era una versión de dos jugadores si no que el jugador interactuaba con él a través de un dial telefónico de rueda.

El primer videojuego que surgió y que cumplía con esos requisitos para considerarse como tal, nace de la mente de *William Higinbotham*[27], físico y gran aficionado al *Pinball*. Éste decidió realizar una variedad de juego, el cual se basaba en un programa de cálculo de trayectorias parabólicas que usaba el ejército americano. El juego se podía visionar en una pantalla circular porque *Higinbotham* lo conectó a un osciloscopio. El resultado de estas invenciones dio lugar al *Tennis for Two*[28].

Consistía en una línea horizontal que representaba el suelo del campo de tenis y de una pequeña línea vertical que simbolizaba la red. Los jugadores elegían el ángulo con el que debía salir la bola y golpearla. Podían participar dos jugadores, los cuales disponían de un controlador con un pulsador que servía para golpear la pelota virtual y un mando analógico que controlaba la dirección de la pelota.

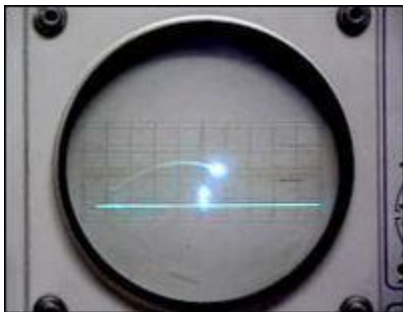


Figura 2. *Tennis for two*

Este juego actuaba de manera bastante más realista que el creado 15 años después, el *Pong*[29], y que gran cantidad de personas lo consideran como el primer videojuego de la historia.

Posteriormente, en 1961 en la ciudad de *Massachussets*, varios miembros del instituto tecnológico se les ocurrieron la idea de crear un juego que mezclara acción con habilidad, el cual se pudiera controlar los objetos moviéndolos por la pantalla. Todo este conjunto de ideas conformó lo que sería el juego *Spacewar*[30].

Figura 3. *Spacewar*

Este juego ocupaba 9K y se presentaban dos naves las cuales se batirían a duelo, un marcador para limitar la duración de la partida y podían llegar al hiperespacio. Éste juego conllevó mucha problemática para salir a delante, aunque finalmente se consiguió en Abril de 1962, tras unas 200 horas de trabajo. *Spacewar* podría considerarse como el primer juego de ordenador de la historia, ya que *Tennis for Two* utilizaba un osciloscopio.

Figura 4. *Computer Space*

En el año 1971 se creó *Computer Space*[31], de las manos de *Nolan Bushnell*[32], que fue fundador de la importante empresa de videojuegos *Atari*[33]. Éste, a partir de una partida al *Spacewar* del que quedó totalmente impresionado, comenzó a crear videojuegos entre los que se destaca *Fox and Geese*. Es un juego en el que había entre cuatro y seis "X", y una "O". Si las "X" conseguían rodear a la "O" ganaban, pero si la "O" comía a alguna de las "X", el triunfo era para el otro jugador.

Bushnell fue muy innovador para su tiempo, debido a que se le ocurrió la idea de que en lugar de jugar en ordenadores, se hiciera en una pantalla con una carcasa con temática del videojuego tapando el cableado, consiguiendo reducir los costes de los juegos. Resultó tan innovadora la idea, que no tuvo éxito.

En la década de los años 70 se produce una clara evolución en los videojuegos. Se destaca a *Ralph Baer*[34] y a su equipo, dado que habían conseguido hacer funcionar en el año 1967 un juego para dos jugadores basado en el deporte del ping pong.

En este juego, al igual que el *Spacewar* y como la mayoría de los primeros juegos de la historia, surgieron problemas a la hora de sacarlo al mercado. Muchas de las empresas por las que preguntó, no les interesaba ese proyecto, hasta que se topó con *Magnavox*. Esta empresa consideró muy interesante la oferta y las condiciones que se establecían, por lo que empieza a trabajar para la comercialización del proyecto.

En 1972 se lanza al mercado *Magnavox Odyssey*[35], siendo por lo tanto la primera consola de la historia, la cual incluía 12 juegos de serie, entre los que se encontraba el ping pong de *Baer*, pero con el inconveniente de que esta consola solo se podía visionar en pantallas y/o televisores de esa misma marca. Meses más tarde, se presentó el primer accesorio de la maquina, un rifle de plástico, con el cual se podía mediante tecnología óptica, interactuar con el juego.

Figura 5. *Magnavox Odyssey*

Como ya se ha comentado con anterioridad, *Bushnell* junto con uno de sus compañeros, *Ted Dabney*[36], fundarían la empresa conocida como *Atari*. Acababa de salir al mercado la *Magnavox Odyssey*, y *Bushnell* ve la consola de esa compañía y el juego del *Ping Pong*, y se le ocurre la idea del desarrollo de un juego similar. Pocos meses después de esa ocurrencia, ya tenían el primer prototipo de éste y por lo tanto, el primer titulo de la recién creada empresa *Atari*, al cual bautizaron con el nombre de *Pong*.

Figura 7. *Pong*

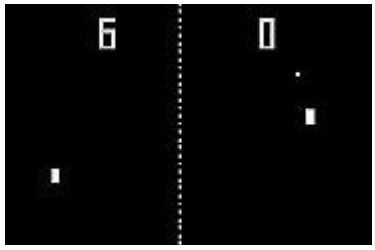


Figura 6. Pantalla de *Pong*

Introducen este nuevo juego en bares y tabernas, causando gran sensación, e incluso las máquinas se estropeaban debido a que las cajas de las monedas estaban a rebosar.
Pong sería el mayor éxito jamás visto hasta ese momento, por lo que *Atari* comienza a crecer de manera espectacular, con lo que con este titulo, la industria de los juegos había nacido definitivamente.

15 nuevas empresas se crearon posteriormente para dedicarse a este mercado, y los juegos que desarrollaron eran simples copias del *Pong*, mientras que *Atari* seguía creando nuevos títulos como *Space Race*, *Rebound* o *Gotcha*, *Quadrapong*… cada uno de los cuales suponía en la práctica la inauguración de un nuevo género.

Años más tarde, en 1974, llegaría el *Home Pong*[37], la primera consola doméstica de *Atari*, la cual permitía jugar al *Pong* en los televisores de las casas. Pero siempre cuesta comercializar estos inventos, como se ha comentado con anterioridad. Finalmente llegan a un acuerdo con una empresa para sacarla al mercado. *Home Pong* se implantó en los hogares y *Atari* estaría en su momento más dulce, con grandes ingresos en las que habían sido vendidas cerca de13.000.000 de unidades.

Figura 8. *Home Pong*

La evolución se empezó a notar en 1975, con la salida del primer juego en color, *Indy 800*[38], que permitía que jugasen simultáneamente 8 jugadores, y como no, perteneciente también a *Atari*.

En 1976 *Atari* crea *Breakout*, el famoso juego de los ladrillos, de *Steve Jobs*[39], que en verdad es la versión de un jugador del famoso *Pong*. Cosechó muchísimo éxito y su creador decidió marcharse de *Atari* para fundar *Apple Computer*[40], por lo que al haber desarrollado este juego, se convertiría en la marca más conocida en el ámbito de la computación.

Figura 9. *Breakout*

En el mismo año llega una máquina capaz de almacenar los juegos en cartuchos intercambiables, siendo esto una clara oportunidad de negocio en el que *Bushnell* tomó nota. Para poder crear una máquina que soportara esto y sacarla al mercado, tuvieron que poner en venta la empresa y *Warner Communications* la compró por 28 millones $.

Figura 10. *VCS/2600*

Con todo este dinero obtenido, *Atari* lanza en 1977 la *VCS/2600*[41], que utilizaría tecnología de microprocesador, cartuchos intercambiables donde se encontraban los juegos almacenados y un gran catálogo de éstos a la venta, que en un principio fueron nueve. El negocio se centraría ahora en la venta de los videojuegos., por lo que se pusieron manos a la obra para el desarrollo de nuevos títulos.

En 1978 llegaría el mítico *Space Invaders* de *Taito*[42]. Un juego que en principio estaba ambientado en una temática bélica, en el que se tenía que disparar a tanques y a aeroplanos. Debido al éxito que obtuvo la película *Star Wars*[43], acabó adoptando su temática definitiva de batalla espacial, destruyendo naves enemigas. Era el primer juego de la historia que se podían guardar las puntuaciones más altas, pero no las iniciales. Fue un éxito absoluto, llegando a tal punto que provocó que el gobierno japonés tuviera que aumentar la producción de los *yens*, por su escasez a causa del juego.

Figura 11. *Space Invaders*

Justo un año más tarde, *Atari* nos presenta el *Asteroids*[44], que al igual que *Space Invaders*, se podía guardar las puntuaciones más altas, pero ahora si las iniciales del jugador, destronando a éste del podio de superventas. Otro gran juego que llegó fue el *Pac Man*[45] no de *Atari*, si no de *Namco*[46]. Supuso una revolución en los videojuegos que se habían desarrollado hasta ahora. Fue tal el éxito y produjo tal impacto que se le dedicó una portada en la revista *Time Magazine*, una serie de dibujos y una canción que causó furor. Se le considera como uno de los videojuegos más influyentes de la historia junto al *Pong*.

Figura 12. *Pacman*

Posteriormente, en la década de los 80, se produce una grave crisis que afectó al sector de los videojuegos en los primeros años de ésta. *Nintendo*[47] empieza a distribuir sus primeras *Game & Watch*[48], las primeras consolas portátiles del mundo siendo precursoras de la *Game Boy*[49], y crearía el *Donkey Kong*[50]. El protagonista se llamaba *Jumpman* el cual tenía que saltar barriles hasta llegar a su Princesa, secuestrada por un mono gigante. A raíz del éxito obtenido por el juego, *Jumpman* cambió su nombre por el de *Mario*, el fontanero más famoso en el mundo de los videojuegos, dado que se convertiría en el protagonista de las sagas de *Mario Bros*[51], invención de *Shigeru Miyamoto*. Este juego solo estaba disponible para la consola que salió en *Japón* llamada *Famicom*, y posteriormente para EEUU, que cambió de nombre pasando a ser *Nintendo Entertainment System*[52]. El juego vendió millones de copias y por lo tanto, también la consola necesaria para poder jugar a este juego. Posteriormente a éste, se desarrollarían títulos tan importantes como es el de *Bomberman*[53].

Figura 13. *DonKey Kong II* en *Game & Watch*

Toda esta cantidad de juegos nuevos que no cesaban de salir al mercado, las réplicas que se producían de ellos por parte de otras empresas del sector, y unido a las decenas de consolas existentes, produjeron que el mercado estuviera saturado. Se tenía un gran número de copias de los juegos en las tiendas, los cuales no se podían vender, teniendo que optar por una estrategia de reducción considerable de los precios y de esta manera poder optar a conseguir algún beneficio. Consecuencia de todo esto, el sector de los videojuegos llegó a una grave crisis.

Esta grave crisis se acabó en el año 1985, produciéndose una recuperación del sector gracias a *Nintendo*. Esta empresa lanza el gran título como es el de *Super Mario Bros*, creando un antes y un después en el mundo de los videojuegos, llegando a vender 10 millones de copias, caracterizado por ser un juego largo y lleno de colorido.

También aparece el mítico juego, *Tetris*[54], de las manos de *Atari*, siendo su origen de *Rusia*. Un juego de puzzles que consiste en encajar piezas de diferentes formas. Es el juego más vendido, conocido y divulgado en toda la historia de los videojuegos.

En Japón, la empresa del sector *Sega*[55], lanzaría la máquina *Master System*, y solo un año más tarde saldría en Estados Unidos, donde no cosecharía tanto éxito como en el país de origen, en el que todavía había gran furor por la *Nintendo Entertainment System*.

La industria sigue recuperándose y aparecen grandes juegos como *Legend of Zelda*, *Arkanoid*, *Castlevania*, *MegaMan*… conformando también los juegos más míticos de la historia.

A finales de esta década, en 1988, aparecerían los famosísimos *Amiga 500*[56] y *Amiga 2000*, de la empresa *Condomore*. Esos modelos eran ordenadores de considerables prestaciones gráficas y también sonoras, con juegos técnicamente muy avanzados. Salió también al mercado la *Mega Drive*[57] en Japón, lo que hizo que *Sega* liderada el mercado mundial con esta maquina de 16 bits, teniendo el titulo de *Sonic the Hedgehog*[58] como referencia, del que se ha obtenido gran éxito durante largos años, y todavía aún.

Figura 14. *Commodore Amiga 500*

Figura 15. *Sega Mega Drive*

En 1989 y del creador de las mencionadas *Game & Watch*, *Gunpei Yokoi*, sale *la Game Boy*, convirtiéndose en una superventas en poco tiempo dentro del mercado de las

consolas portátiles, debido a su reducido precio y al amplio catálogo de juegos disponibles, todos ellos en cartuchos. La consola ha ido cambiando a lo largo de los años, incorporando eventualmente la capacidad de ejecutar juegos en color, reducción de tamaño…También tuvo sus imitaciones como la *Lynx* de *Atari*.

Figura 16. *Nintendo Game Boy*

La década de los 90 se podría considerar como la década dorada de los videojuegos. La causa de que los videojuegos estén tan expandidos y sean tan populares en nuestros días, es en parte por esta década.

Las videoconsolas dieron un importante salto técnico gracias a la competición que surgió entre las maquinas de 16 bits. Aparece la *SNES, Super Nintendo Entertainment System*, consola que conseguiría enormes ventas en todo el mundo, gracias en parte a su antecesora y su legado. También se caracterizaría por una de las máximas rivalidades que dicen que ha existido en la historia de los videojuegos por la competencia con la *Mega Drive*.

Figura 17. *Super Nintendo*

Seguidamente *Sega* creó su consola portátil más famosa, la *Game Gear*[59], que llevaba una CPU a 3,58 Mhz con 8 Kb de RAM. No pudo hacer sombra al éxito comercial de *Game Boy*, porque además de no conseguir tanto auge, tenia una seria de factores negativos como que tenía muy poca duración de batería y era de un tamaño más grande que la portátil de *Nintendo*. También sacó en Estados Unidos la *Sega Master System II*[60], una consola más pequeña y ligera, pero no cosechó gran éxito debido a que el cambio de generación de consolas ya se había producido.

También se nos presenta la *Neo Geo*[61] de *SNK*, un aparato muy avanzado para su época, en el que se podía jugar a los títulos de las máquinas recreativas en tu casa. Era de precio elevado por lo que mucha gente no podía tenerlo.

En los años posteriores, surgieron gran cantidad de títulos, que a día de hoy, se consideran clásicos como son: *Mortal Kombat*, *Wolfenstein 3D*, *Alone in the Dark*, *Doom* y *FIFA*.

Rápidamente los videojuegos en 3D fueron ocupando un importante lugar en el mercado, principalmente gracias a la llamada "*generación de 32 bits*". En 1994 *Sega* sacó la consola *Saturn*[62] en terreno nipón, llegando a vender 170.000 unidades el primer día que salió a la venta. *Sony* estaba a punto de sacar su nueva *PlayStation*[63], lo que provocó que *Sega* tuviera que rediseñar su consola, por lo que hubo muy pocos títulos en su lanzamiento, y además de los que se podían disponer, se acabaron con prisas. Con la *Play Station*, *Saturn* fue perdiendo terreno y, posteriormente, con la llegada de *Nintendo 64* quedaría relegada a un tercer lugar en la lucha por el mercado.

Es interesante saber que la consola de *Sony*, iba a ser un proyecto conjunto con *Nintendo*, pero éste rechazó la oferta de *Sony* debido a que las consolas antecesoras con lector de CD no habían cosechado éxito alguno. Por lo tanto, *Sony* decidió lanzar su proyecto de manera independiente con el nombre de *PlayStation*. Esta decisión fue la más acertada porque triunfarían de la manera más espectacular y conformaría un gran punto de inflexión en este mundo. Debutaban en el mercado de las videoconsolas y barrían a toda su competencia, convirtiéndose en una de las videoconsolas más vendidas de todos los tiempos.

Se debe mencionar las sagas que se encontraban en su catálogo, y que gente de mi generación, hemos disfrutado mucho de ellos: *Metal Gear Solid*, *Resident Evil*, *Tomb Raider*, *Tekken*, *Gran Turismo*, *ISS Pro Evolution*, *Oddworld: Abe's Oddysee*, *Medievil*, *Final Fantasy* o *Crash Bandicoot*.

Figura 18. *Sega Saturn*

Figura 19. *Nintendo 64*

Figura 20. *Sony Playstation*

En 1996 llega la *Nintendo 64*, que quedó en segundo lugar en el mercado de las consolas de sobremesa, tuvo buenas ventas y presentaba un gran catálogo de juegos con mucha calidad. Tenía un CPU de 64 bits a 93,75 Mhz, y grandes innovaciones como el mando de control, siendo el primero en llevar unos botones dispuestos en forma de cruz y la vibración incorporada. Dentro de su catálogo se destacan entre otros: *Super Mario 64*, *The Legend of Zelda: Ocarina of Time*, *GoldenEye 007*, *Perfect Dark*, *Mario Kart 64*, *The Legend of Zelda: Majora's Mask*…

Al año siguiente, *Nokia*[64] presenta los primeros teléfonos móviles con sistema operativo y capacidad de ejecutar *Java*. Decidió ofrecer algún tipo de entretenimiento en esos pequeños dispositivos que tenían botones y una pantalla LCD en blanco y negro. Desarrolló el juego *Snakes*[65], el cual llegó a ser uno de los juegos más expandidos dentro de este tipo de dispositivos. El objetivo consistía en comer unas

bolas que provocaba que la cola de la serpiente protagonista creciera, y a su vez, al estar dentro de un laberinto, intentar no chocar con las paredes u obstáculos existentes.

Posteriormente un año más tarde, salió al mercado nipón la última consola de *Sega*, la *Dreamcast*[66], que posteriormente lo haría en EEUU y Europa. Contaba con una CPU a 200 Mhz y utilizaba como soporte el GD-ROM y era la primera consola de 128 bits. Ésta venía marcada por el fracaso de *Saturn* y temía un gran rival que estaba a punto de salir, la *PlayStation 2*. Pese a ser una buena consola acabó fracasando y con ella *Sega* ponía punto y final a su trayectoria en las consolas de sobremesa y portátiles.

Pasamos ahora a la era de los videojuegos en la que nos encontramos. Se caracteriza por las consolas con gráficos HD impresionantes, posibilidad de almacenamiento interno por su disco duro integrado, de conexión a Internet y sobre todo por la cantidad de juegos que ofrece su catálogo. Este sector se ha transformado en una industria multimillonaria.

Todo comienza en el año 2000, con la aparición de la consola más vendida de la historia por la empresa *Sony: PlayStation 2*. Era una maquina de 128 bits, la cual tuvo serios problemas para distribuirse en sus inicios por sus problemas de sobrecalentamiento y de hardware. También presentaría la *PsOne*, un rediseño de la primera consola que sacó, y que se caracterizaba por reducir su tamaño.

En 2001 aparece una nueva consola de *Nintendo*, la *GameCube*, que no consiguió atraer al público con un gran fracaso comercial, y que desde entonces se dedicó más a los *handhelds*. A ella se le suma la *Xbox*[67], que suponía la entrada al mercado por parte de *Microsoft*[68], con características similares a la maquina de *Sony* y con grandes títulos como *Halo*[69] en su catálogo. Estas dos consolas, no llegaron a alcanzar el éxito que consiguió *PlayStation 2*.

Figura 21. *Sony Playstation 2*

En 2001, la compañía *Nintendo* con su nueva estrategia de mercado, saca en el mercado de los handhelds, la *GameBoy Advance*, pero no conseguirá gran éxito ni recuperar el mercado perdido que había desperdiciado la consola *GameCube*.

En el 2003 *Nintendo* presenta el rediseño de la *GBA*, y por otro lado, la compañía finlandesa *Nokia*, se atreve con el mundo de las videoconsolas portátiles y lanza su móvil- consola *N-Gage*[70]. Tenía un procesador a 104 Mhz, reproductor MP3, de video, radio FM integrada etc. Todo aparentaba que iría bien, pero su elevado precio, provocó que no consiguiera tal éxito.

Un año después, *Nintendo* lanza al mercado, consiguiendo ser una de las más vendidas de la historia, la consola portátil *Nintendo DS*. Sigue siendo en nuestros días, uno de los productos con más ventas en el mundo. *Sony* también desarrollo su consola portátil *PSP*, pero no consiguió superar las expectativas que se esperaban de ella, ni las ventas de la *Nintendo DS*. Junto a *PSP*, durante este año, *Sony* también presenta el rediseño de *PlayStation 2*, *PsTwo*, también de reducido tamaño.

Figura 22. *Nintendo DS*

Figura 23. *PSP*

En 2005, es el año donde surgen las consolas de la siguiente generación "*Nextgen*", con la aparición de la *Xbox 360*, modelo mejorado del anterior pero con continuos problemas hardware. *Sony*, pocos meses después, responde sacando la nueva *PlayStation 3*, con poca variedad de juegos y un elevado precio, que aun todavía sigue siendo así. Y faltaba *Nintendo* con la consola *Wii* en 2006. una máquina que se caracteriza por su innovador sistema de control por movimiento y unos juegos sencillos y de pobre calidad grafica, que hizo que *Nintendo* volviera al mercado de las consolas de sobremesa, poniendo de nuevo a la compañía en el lugar que le correspondía dentro de la historia de los videojuegos.

Figura 24. *PlayStation 3*

Figura 25. *Xbox 360*

Las compañías continuaron lanzando títulos, y muchas suponían grandes innovaciones, como es el caso de *Guitar Hero*. Es un juego que con un original sistema de control, parece que formas parte de una banda de rock, pudiendo tocar la batería y guitarra. Esto dio lugar a una saga con más títulos de los que se venden grandes cantidades de copias.

The Sims[71] en el año 2000 crean una nueva modalidad de juego como es de la simulación social, pudiendo crear tu propio barrio, tipos de casas, vecinos, personajes y poder interactuar entre ellos.

A partir de entonces se da con la idea de ofrecer libertad de movimientos al personaje y de acción, y unirlos con unos argumentos más complejos. *Grand Theft Auto* es claro ejemplo de esto, consiguiendo un éxito rotundo, formando una gran saga.

Grandes títulos de esta década que se deben destacar son: *PES*, *Resident Evil*, *Halo*, *Prince of Persia*, *Call of Duty*, *Medal of Honor*, *Metal Gear Solid*…

Ha habido pocas referencias en cuanto a juegos en los teléfonos móviles. Éstos han ido evolucionando conforme lo hacían los terminales. Se caracterizaban por ser juegos de un solo color, reproducir sonidos polifónicos, ser bastantes sencillos, incluirse en la memoria ROM del móvil, utilizar la pulsación del teclado, y han evolucionado hacia juegos de color, con sonidos multimedia, posibilidad de tener varios juegos pudiendo ser borrados, pantallas táctiles y gran cantidad de posibilidades.

Nokia ha sido la clara precursora en la historia de los videojuegos de los teléfonos móviles. Con la inclusión de *Java* en sus terminales, algunas empresas como la francesa *Gameloft*[72] desarrollaron videojuegos en blanco y negro para esas pequeñas pantallas y resultaron un éxito, además de que con *Java* se abrió grandes posibilidades al mercado en este sector. Pero no es hasta la época donde nos encontramos cuando se origina una gran revolución.

Se producen grandes avances en la programación, creación de nuevos sistemas operativos, un rápido crecimiento del sector y dispositivos cada vez más potentes, permitiendo ejecutar videojuegos de gran complejidad. Los smartphones son el claro ejemplo de la reunión de las características de los dispositivos que se han inventado. Permiten en pequeño tamaño, ejecutar juegos de gran calidad gráfica y motora.

Actualmente el mercado de los videojuegos para móviles es más grande que cualquier otro mercado de videojuegos portátiles, obteniendo cifras de ventas muy elevadas. Como aplicaciones para el futuro se esperan videojuegos en 3D y videojuegos en red a través de teléfono o *Wifi* o *Bluetooth*.

2.2 Videojuegos en *Android*

El sector de los juegos móviles se disparó desde el lanzamiento del *iPhone* y más si cabe con el *iPad*. El más claro ejemplo es el famoso y conocido juego *Angry Birds*[73], el cual está disponible prácticamente para todos los sistemas operativos móviles y ha conseguido más de 200 millones de descargas. Con todo esto se prevé, según un estudio de *IDC*, que en 2015 habrá 182.700 millones de descargas de apps al año, y los juegos serán el tipo de aplicación más descargada del mercado[74].

Pero no hace falta irse tan lejos en el tiempo, si no que en este año ya se puede observar este ritmo de crecimiento, observando y analizando que el número de aplicaciones y juegos acumulados en *Android Market* es superior a las 500.000, en la siguiente gráfica[75].

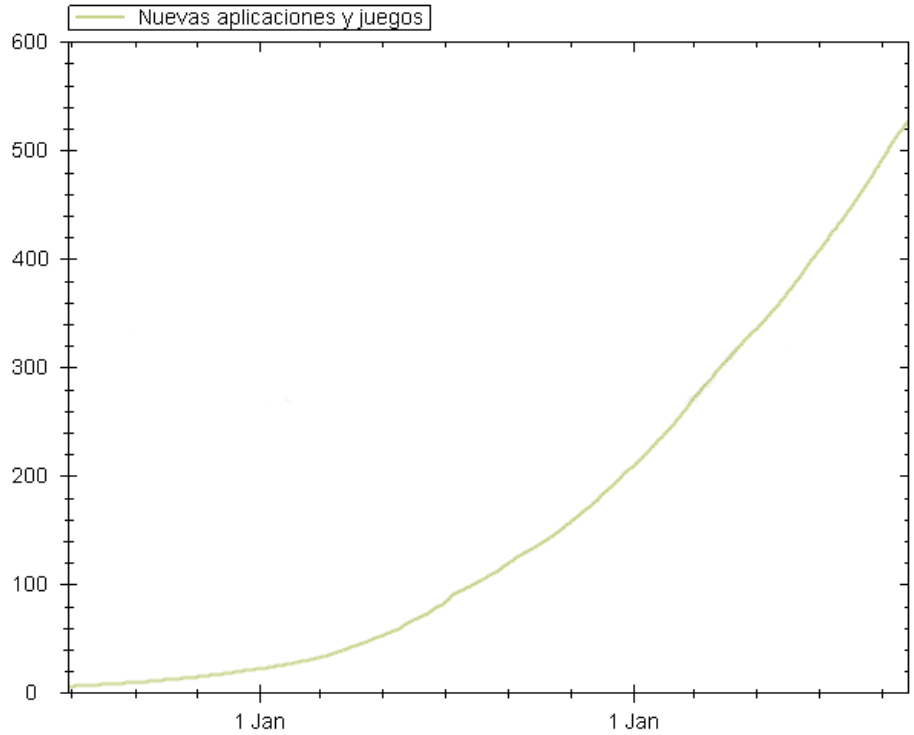


Figura 26. Gráfica número de aplicaciones acumuladas en *Android Market*

También en la misma fuente, se puede observar el número de aplicaciones nuevas que se publican casi de forma diaria por parte de los desarrolladores en la tienda de software en línea de *Google*.

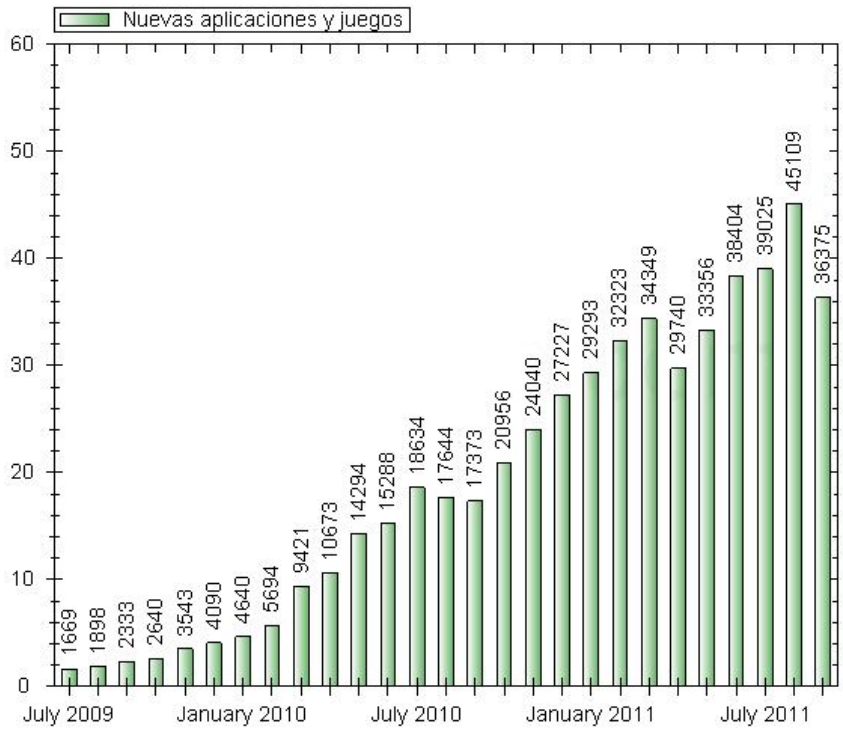


Figura 27. Gráfica número de aplicaciones nuevas por mes en *Android Market*

Dentro del número de aplicaciones que se ha podido observar en *Android Market*, la más popular son los juegos, con un 26% del mercado y los cuales serían el único tipo de

aplicación por las que los usuarios estarían dispuestos a pagar. En segundo lugar se encuentran las apps de utilidades con un 16%, al igual que las de entretenimiento, información según datos de *Chomp* de Agosto 2011[76].

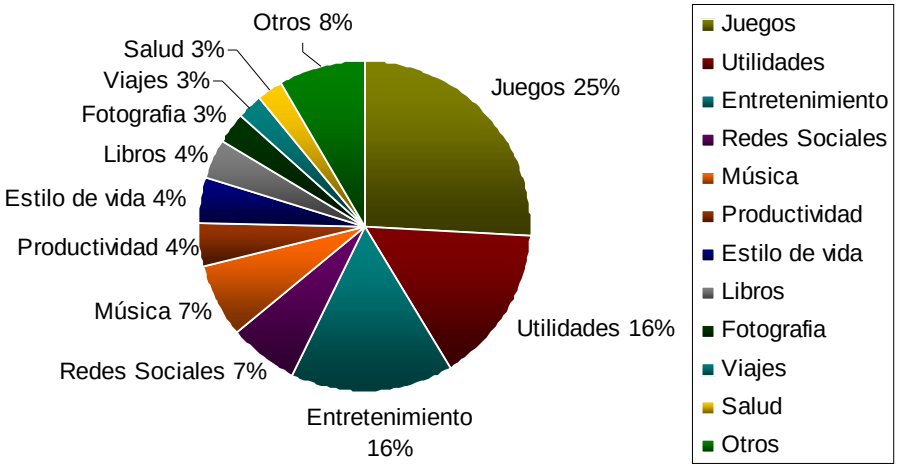


Figura 28. Gráfica categorías de aplicaciones más populares

Pero las empresas de desarrollo de videojuegos para móviles, en la mayoría de videojuegos se programan primero para *iPhone*, y meses o un año más tarde para *Android*, y con la salida al mercado del *iPhone 4*, ha hecho que se más difícil portar esos juegos a *Android*, pero como se analizará más adelante, esta situación va a cambiar.

Las razones de por qué se tarda tanto en sacar la versión del videojuego para *Android*, o por qué ni siquiera se llega a desarrollar son tres:

1. **La fragmentación del sistema:** existen muchas y distintas versiones liberadas por *Google*, desde la 1.5 hasta la 2.3.4. Todas estas se encuentran en el mercado en los dispositivos móviles, y se tienen que tratar de distinta manera a la hora de desarrollar aplicaciones. Existen diferencias entre versiones, pero hay cierta compatibilidad hacia atrás entre ellas, por lo que todo lo que se haga sobre la 1.5, valdrá para las posteriores. Por lo tanto, esta situación a las empresas las desconcierta debido al gran número de versiones de *Android*.

   Pero no solo la fragmentación que existe en el mercado de las versiones de *Android*, si no que además de esto, se tiene que tener en cuenta las variaciones de cada operadora para cada dispositivo, porque puede existir importantes variaciones de rendimiento y de memoria entregada, dependiendo de qué haya hecho la operadora con él antes de ponerlo a la venta, además del gran número de dispositivos móviles existentes con *Android*. Esto no pasa en el de *Apple*, dado que solo hay uno y en el que las empresas desarrolladoras de videojuegos ya saben que memoria, rendimiento… les va a entregar el dispositivo en cada momento.

   Una posible solución a este problema de fragmentación del sistema, sería el desarrollo de un sistema de actualizaciones más eficiente, común para todos los dispositivos móviles, en el que las distintas operadoras no pudieran realizar modificaciones de ninguna índole y que generara muchos menos problemas a las

empresas, como *Epic Games*[77] que ya se ha retirado del desarrollo para *Android*. Si no se produce alguna mejora, no se ampliará el mercado de videojuegos en *Android*, al contrario de como lo esta haciendo su mayor competidor.

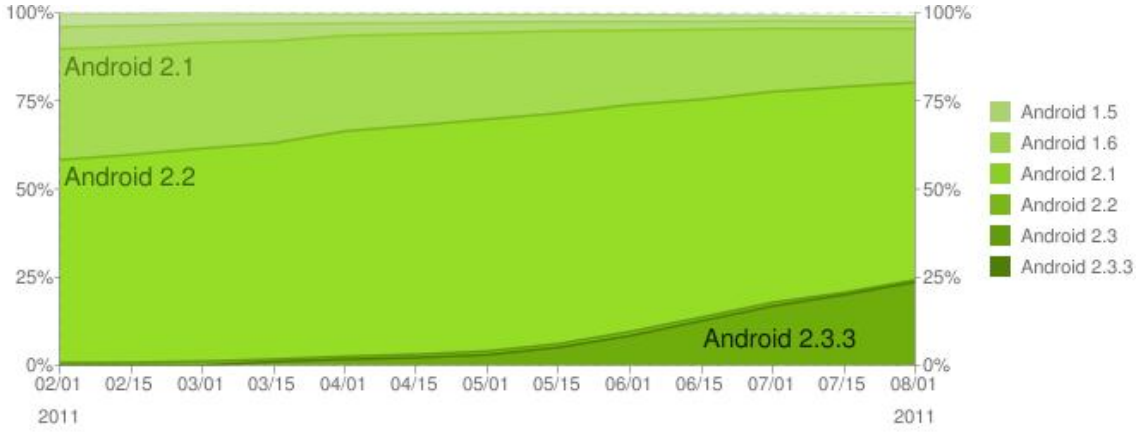


Figura 29. Gráfica fragmentación de los dispositivos *Android* en el mercado

2. **La rentabilidad:** es uno de de los aspectos que ha posicionado a *iOS* como una importante plataforma. No quiere decir que no existan personas que mediante la publicación de sus aplicaciones en *Android Market* no hayan conseguido dinero, al contrario, pero las ganancias que genera la *AppStore*[78] son mucho mayores, dado que hace poco *Apple* anunció que ha logrado los 15.000 millones de aplicaciones descargadas.

   Esto es debido a la poca rentabilidad de los juegos para *Android*, muchos de ellos gratuitos y que se financian mediante publicidad. Se puede observar a continuación, las aplicaciones descargadas en cada uno de los sistemas operativos, siendo datos de Agosto de 2011 ofrecidos por *Chomp*[79]

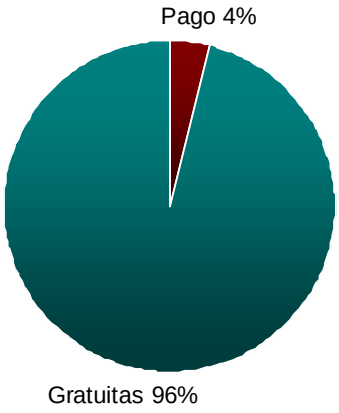


Figura 30. Aplicaciones descargadas: de pago vs. gratuitas en *Android*

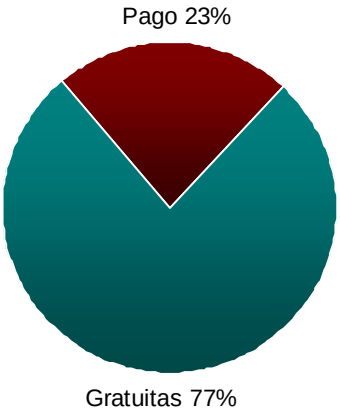


Figura 31. Aplicaciones descargadas: de pago vs. gratuitas en *iOS*

3. **Las pantallas:** los dispositivos móviles con sistema operativo *Android*, tienen unas pantallas que dejan mucho que desear, consumen mucha batería, el multitouch solo está accesible a partir de la versión de *Android 2.2 Froyo* y no consiguen llegar a la calidad gráfica necesaria. *Samsung* es la empresa que está poniendo más empeño en el desarrollo de mejores pantallas, y que dentro de

poco, sacaran móviles con pantallas *Super Amoled*[80], y de este modo conseguir equipararse al modelo de *Apple*.

A pesar de todos estos aspectos negativos, poco a poco se está comprobando que juegos de *iPhone* están saliendo también para *Android*. Esto es debido a que los fabricantes de móviles y procesadores quieren ofrecer la mejor experiencia posible, y han conseguido aumentar la potencia en los móviles, porque sin ésta, pocos juegos buenos se sacarán al mercado. Consecuentemente, se ha de tener un móvil de gama alta con *Android* porque la mayoría de los juegos están siendo optimizados para los procesadores más potentes.

El dispositivo móvil de mayor potencia de referencia del sistema operativo *Android* para las empresas de desarrollo de videojuegos móviles, es el *Galaxy S2*[81]. Gracias a éste, desarrolladores y empresas como *Gameloft*, han empezado a traer sus juegos a *Android*.

Por lo tanto, la situación actual de los videojuegos en el sistema operativo *Android* es de leve atraso. Hay una gran variedad de juegos, pero la calidad no es comparable a la que se obtiene sobre la plataforma de *Apple*, la potencia de los dispositivos móviles es inferior, exceptuando determinados casos, y no resulta muy rentable a los desarrolladores crear aplicaciones en ese sistema. También existe la negativa por parte de importantes empresas desarrolladoras de juegos y que *Google* niega los problemas de fragmentación que se han citado con anterioridad, por lo que una solución a esos problemas no se desarrollará.

Pero el futuro es de *Android*. A día de hoy es el único sistema operativo que ha plantado cara a *iOS*, obteniendo el 40% del mercado mundial. Existen empresas como *Gameloft* que desarrollan sus asombrosos juegos para ellos, pero falta confianza y un salto de calidad en *Android*, que se dará en un futuro cercano, para comparar sus videojuegos a los de *IPhone*.

2.2.1 Tipos de videojuegos

Se ha analizado y observado que los videojuegos son el tipo de aplicación favorita en *Android Market* por parte de los usuarios, pero dentro de este tipo, existen variedades de géneros, que son:

- **Arcade y acción:** implica manipular un personaje a través de una serie de niveles de dificultad progresiva pero no se produce el desarrollo de una historia o el de los personajes. Juegos como: *Angry Birds*, *Robo Defense*, *Angry Birds Rio*, *Fruit Ninja*…

- **Carreras:** ambientados en la competencia de coches, motocicletas y cualquier vehículo. Los más famosos son: *Drag Racing*, *Trial Xtreme*, *Racing Thunder*, *GT Racing*…

- **Casuales:** juegos dirigidos o utilizados por gran número de jugadores pero ocasionales. Pueden pertenecer a cualquier género y se caracterizan por sus sencillas reglas. Son de este tipo: *Jewels*, *Paper Toss*, *Air Control*, *Abduction*…

- **Deportes:** basados en cualquier tipo de deporte usando las reglas características de éste. Títulos de esta variedad de juego son: *Billar Maestro Poll*, *Soccer*, *Sniper*…

- **Juegos de cartas y casino:** ambientados en cualquier juego de cartas existente y empleando las reglas de cada uno. *Live Holdem Poker Pro*, *Solitaire*, *Super Sudoku*…

- **Puzzles y juegos para ejercitar la mente:** en ellos se exige cierta habilidad mental al jugador e involucra problemas de lógica, estrategia…Títulos de este género son: *Bubble Blast 2*, *Mouse Trap*, *Wordfeud*…

Se observa que se pueden elegir gran número de juegos, pero existen algunos que resaltan de entre los demás y son además favoritos en todos los sistemas operativos, ya sea por tener un rasgo novedoso, por enganchar a los usuarios o por ser clásicos dentro de la historia del videojuego. En *Android*, al igual que en otros sistemas operativos, se tienen que dividir los juegos en aquellos que su descarga es gratuita de los que se tienen que pagar, debido a que aquellos juegos que son gratuitos tendrán muchas más descargas que los de pago, y más en *Android*, que todas las aplicaciones que se descargan solo un 4% son de pago, como se ha visto con anterioridad.

A continuación, se realiza un análisis de los juegos más descargados dentro de los de pago y los gratuitos, incluyendo el por qué de su auge y las descargas realizadas por parte de los usuarios. Información proporcionada por *Android Market*.

2.2.1.1 Gratuitos

Figura 32. *Angry Birds*

Angry Birds es el juego por excelencia de *Android*. Su rasgo más característico es el uso de un tirachinas para lanzar los pájaros protagonistas con distintos tipos de habilidades contra sus enemigos con el fin de eliminarles y recuperar sus huevos.

Descargas totales a Julio del 2011:
10.000.000 - 50.000.000

Figura 33. *Drag Racing*

Drag Racing es un juego de carreras con el mejor control real. Se caracteriza por tener un gran número de coches disponibles y ser multijugador, de esta manera se pueden retar a otros usuarios y comprobar cuál es el mejor coche dentro de la comunidad del juego.

Descargas totales a Julio del 2011:
5.000.000 - 10.000.000

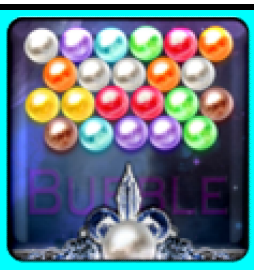


Figura 34. *Shoot Bubble*

Shoot Bubble es el juego en el que se disparan y revientan burbujas. Se tiene que realizar una combinación de tres burbujas del mismo color para reventarlas y se caracteriza por su adicción y ser un clásico.

Descargas totales a Julio del 2011:
1.000.000 - 5.000.000

El juego más popular del género de plataformas y dentro de la sección de gratuitos es:

Figura 35. *Droid Jump*

Extrema Droid Jump es un sencillo y adictivo juego de plataformas. Su rasgo más característico es el uso del sensor de orientación para guiar al androide en su ascensión en el escenario, apoyándose sobre las plataformas existentes que le proporcionan impulso.

Descargas totales a Julio del 2011:
1.000.000 - 5.000.000

2.2.1.2 De Pago

Figura 36. *Cut the Rope*

En *Cut the Rope* se deben cortar las cuerdas que se disponen en el escenario, para alimentar al protagonista, siendo muy novedosa la idea. Viene de *iOS* donde cosechó gran fama, colocándole en el 1º puesto de aplicaciones más descargadas de la *AppStore*.

Descargas totales a Julio del 2011:
100.000 - 500.000

Figura 37. *Spirit HD*

Spirit HD es un juego retro inspirado en el caos arcade, donde se toma el control de *Spirit* el cual se mueve mediante el deslizamiento del dedo sobre la pantalla. Existen distintos tipos de enemigos a los que se les tienen que forzar para abandonar el espacio.

Descargas totales a Julio del 2011:
10.000 - 50.000

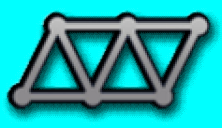


Figura 38. *X Construction*

En *X Construction* se tiene que construir un puente con el número de vigas de acero que se proporcionan, para que el tren pueda cruzar el valle seguro. Es una idea novedosa y que engancha.

Descargas totales a Julio del 2011:
100.000 - 500.000

El juego más popular del género de plataformas y dentro de la sección de pago es:

Figura 39. *Squibble*

Squibble es un juego de plataformas 2D. Se ha de desplazar al personaje mediante el uso de sus 2 tentáculos, para avanzar, trepar, descender, balancearse y usarlos como tirachinas para saltar, resultando esta idea muy novedosa y original.

Descargas totales a Julio del 2011:
10.000 - 50.000

2.3 Herramientas de desarrollo

Existen distintos tipos de herramientas para el desarrollo de aplicaciones, en las que a continuación, se describen dos de las que actualmente más se utilizan, y otras dos que se emplearán en un futuro cercano, los cuales ampliarán las posibilidades de *Android* en los videojuegos de manera considerable.

La primera opción escogida para programar aplicaciones en *Android* es la de instalar el kit de *SDK* de *Android* y el programa *Eclipse*. El *SDK* es una serie de herramientas de desarrollo de software *Android* que funciona tanto en *Windows*, *Linux* o *Mac*, y *Eclipse* sirve para programar en *Android*, siendo una de las herramientas de desarrollo *Java* más versátiles del momento. Además de esas dos herramientas anteriormente citadas, se tendrá que tener instalado *Java*, puesto que hacen uso de él así como cada programa creado hará uso de este lenguaje.

Google también ha incentivado recientemente el desarrollo de juegos, gracias a su *NDK*[82], *Native Development Kit*, que ofrece a los desarrolladores programar directamente en código nativo, donde se le sacará más potencia al hardware de los dispositivos móviles saltando la maquina virtual *Dalvik* de *Android*.

Por otro lado, en el próximo año 2012 se unirán más herramientas para el desarrollo de juegos en *Android* por parte de grandes empresas, facilitando la creación de juegos para

distintas plataformas en *Android* y portar los juegos ya existentes de otros sistemas operativos. Los programas que facilitarán todo esto serán *Havok*[83] y *Unity*[84].

2.3.1 Havok

Figura 40. Símbolo de *Havok*

Havok, de origen irlandés, se conoce por desarrollar motores de físicas muy potentes y relacionadas con el comportamiento de ciertos objetos ante ciertas acciones de muy buena calidad, muy usado por la industria del videojuego. Está siendo adaptado actualmente al sistema *Android* y especialmente para el *Sony Ericsson Xperia Play*[85], y posiblemente se extienda a otros terminales de gama alta.

Con este programa se muestra la clara evolución de los videojuegos hacia una mayor potencia y rendimiento, con la consecuente mejora que se producirá en los dispositivos.

2.3.2 Unity

Figura 41. Símbolo de *Unity*

Para facilitar portar los juegos de *iPhone* a *Android* llega *Unity*, de la empresa *Unity Technologies*. Es una herramienta de programación multiplataforma, que según la información, en un par de semanas es posible portar un juego de *iPhone* a *Android*, incluso darse el caso de solo tener que cambiar de plataforma dentro del programa y darle al botón de compilar. También sirve para la programación, obteniéndose muy buenos resultados.

A esta iniciativa ya se han sumado más de 50 desarrolladores que han portado sus títulos gracias a este software, y además se ha lanzado una versión más factible que pasa a costar 400$. Varias empresas ya lo han utilizado y han tardado como máximo 2 semanas en hacer la portabilidad. Ya no es un impedimento para los desarrolladores que quieran mejorar sus ventas, pasando sus aplicaciones a la plataforma de *Google*.

Esta herramienta es muy buena noticia para el mundo de *Android*, ya que grandes títulos que se encuentran en la *AppStore*, por fin serán portados al sistema operativo de

Android pudiendo disfrutar de ellos aumentando las posibilidades de este mercado a *Android*.

2.3.3 Opción escogida

La opción escogida para el desarrollo del videojuego del presente proyecto, es la de programar utilizando el *SDK* de *Android* más *Eclipse*, anteriormente mencionados, además del uso de una liberaría de otro PFC que está especialmente diseñada para realizar interfaces 3D sobre videojuegos de plataformas como el que quería desarrollar.

Por el contrario *Havok* y *Unity* son programas más generales, muy potentes y por tanto más complicados de usar además de ser comerciales, dirigidos a grandes empresas que puedan pagar el importe de sus licencias y con el inconveniente de que todavía no se encuentran disponibles.

2.4 Futuro de los videojuegos en *Android*

Como se ha comentado con anterioridad, *Android* se ha estado quedando atrás con respecto a la plataforma de *Apple* en cuanto al desarrollo de videojuegos se refiere, no ha habido una evolución.

Epic Games ha desarrollado un *port* del motor gráfico *Unreal Engine*[86] a *iOS*, que permite desarrollar juegos increíbles por optimizar el proceso productivo, obteniendo por esto, gran importancia en el sector de videojuegos móviles. Esta empresa ya rechazó desarrollar juegos para *Android*, por lo que si ya existía diferencia en los juegos entre ambos sistemas operativos, ahora hay más.

A pesar de esto, la gran empresa de desarrollo de videojuegos *GameLoft*, ha comprado una licencia del motor gráfico *Unreal Engine* a *Epic Games*, con intención de comenzar a programar videojuegos para *Android*. Esto es una buena noticia porque supone que esta empresa se mete de lleno en el desarrollo de videojuegos del sistema operativo de *Google* y se dará un salto de calidad en ellos.

Figura 42. Símbolo de *Nvidia Tegra II*

Además de lo anterior, *Android* contará con un catálogo de juegos respetable debido a la empresa de tarjetas gráficas *Nvidia*[87] que es la desarrolladora de los potentes procesadores de doble núcleo *Nvidia Tegra 2* para los smartphones, con su distintivo *THD*, que ha anunciado que creará su propio *Market* en donde se distribuirán versiones

mejoradas de videojuegos, pero optimizados para el procesador citado con anterioridad.

Otra importante empresa que tampoco quiere quedarse atrás es *Qualcomm*, que desarrolla procesadores también para smartphones. Ha conseguido un acuerdo con la desarrolladora de juegos *Gameloft*, en el que determinados títulos van a ser optimizados para los procesadores que contiene el *HTC Desire HD* y el doble núcleo MSM8×60.

Otro aspecto positivo para el futuro de *Android* y sus videojuegos, es el dispositivo móvil *Sony Ericsson Xperia Play*, y todo lo que este terminal supone**.** Este dispositivo que ya está en el mercado, es el primer smartphone y consola reunidos en un solo terminal, por lo que *Sony* apuesta por este tipo de dispositivos. Pero además de ser un terminal muy potente, supone también que *GameLoft*, esa empresa que ha comprado el motor *Unreal Engine* a *Epic Games*, haya lanzado 10 juegos para *Xperia Play*, y que importantes empresas como *EA*[88] o *Codemasters*[89], desarrollarán juegos para este dispositivo móvil.

Figura 43. *Sony Xperia Play*

También este dispositivo móvil conlleva más aspectos. *Sony* ha decidido que los juegos que se supone que serian propios del *Xperia Play*, también estarán disponibles para los demás dispositivos de otras marcas como smartphones, tablets… Esto será posible gracias a la iniciativa *PlayStation Suite*[90] y a su tienda *PlayStation Store*[91], en la que se podrá disfruta de un amplio catalogo de juegos, desde clásicos de la primera *PlayStation* hasta nuevos títulos.

En conclusión, el dispositivo móvil *Xperia Play* ha creado una revolución en *Android*, porque además de que en un futuro volvamos a los teclados para los smartphones, las empresas y desarrolladores ven en *Android*, una buena opción de futuro a explotar. *Android* se encuentra en una etapa de transición y evolución, en la que puede que dentro de un año, la confrontación entre los dos grandes sistemas operativos en el ámbito de los videojuegos, se pueda considerar equiparado e incluso estar por encima de *iOS*. Hasta que llegue ese momento, *Android* dispone del *Xperia Play* como referente en potencia y calidad.

Capítulo 3

Descripción del videojuego

3.1 Descripción del videojuego

Con la unión de dos proyectos fin de carrera produce como resultado el desarrollo de un videojuego del género plataformas sobre el sistema operativo de dispositivos móviles *Android*.

Este tipo de videojuegos se caracterizan por que el usuario controla a un personaje que debe avanzar por el escenario ya sea caminando, corriendo o saltando, encontrándose con una serie de enemigos que debe de esquivar, mientras se recogen objetos para poder completar el juego. El desplazamiento de los personajes, en general, tiene un desarrollo horizontal, por lo que se suelen usar vistas de desplazamiento horizontal hacia la izquierda o hacia la derecha.

Cube´s War es el nombre del videojuego desarrollado, cumple con las características anteriormente descritas, pero con la salvedad de que se utilizan gráficos 3D. Esto permite que en los escenarios desarrollados no solo tengan componentes en los ejes X e Y, sino que también se pueden añadir elementos en el eje Z, por lo que amplían la posibilidad de desplazamiento del personaje una coordenada más, avanzando en profundidad.

Son tres los escenarios de este tipo con los que se encuentra el usuario, por los que debe hacer avanzar al personaje. Tienen una dificultad media y cada uno de ellos está ambientado en un lugar distinto, como son en un bosque, en el glaciar y sobre la superficie lunar.

El usuario mediante el manejo del personaje con la pulsación de los botones que componen el pad y el de *Saltar* dispuestos en la pantalla, avanza por el escenario para poder completar éste, habiendo recogido con anterioridad la llave dispuesta en el escenario que le abrirán las puertas al siguiente nivel. Cuando se complete el tercer escenario, se habrá superado la totalidad del videojuego, por lo que se volverá a iniciar éste desde el primero.

Durante el progreso del usuario, se encontrará con distintos elementos que le impedirán completar éste cometido. Uno de ellos es que en el escenario por el que discurre se acabe bajo sus pies, por lo que puede elegir dirigirse hacia otro camino que exista para seguir avanzando, o decidir saltar ese vacío hacia la plataforma siguiente y continuar con el progreso.

Otros impedimentos es el de la aparición de piedras que caen de la parte superior del escenario de forma cíclica, en el que el usuario debe intentar no colisionar con ellos para no ser eliminado.

También existen elementos de forma puntiaguda sobre los que el usuario con su avance, no debe dejarse caer o saltar sobre ellos, porque será eliminado y tendrá que comenzar el nivel desde el principio.

Los enemigos son los componentes que pondrán las cosas más difíciles al jugador, dado que perseguirán al personaje que controla a donde quiera que vaya del escenario, siempre que no se acabe la plataforma de avance bajo sus pies. Existen tres tipos de enemigos: el primero avanza de izquierda a derecha con una velocidad lenta, el segundo un poco más rápido que el anterior, y el tercero es el más peligroso, dado que aunque avance con la misma velocidad en los ejes que el primero, también puede avanzar sobre el escenario en el Z, por lo que podrá progresar más en él. El usuario los podrá distinguir por su distinta apariencia.

Debido a la cantidad de sensores que disponen los dispositivos móviles, mediante el empleo de uno de ellos, se consigue que el movimiento que realice el usuario del terminal, mueva el entorno 3D del juego en el sentido en el éste lo haga. Esto permite que se pueda visualizar si en el escenario existe plataforma en el eje Z deseado, y de esta forma, mover o no al personaje controlado hacia esa dirección.

En todo momento el usuario podrá pausar la partida y decidir si sale de ésta para encaminarse al menú principal. Si es así, se visualizará éste con las distintas opciones que alberga el videojuego. Una de ellas es la posibilidad de observar los creadores que han desarrollado esta aplicación, otra es la de modificar las opciones del juego como son la activación de la reproducción de efectos de sonido y de la vibración del dispositivo móvil durante el transcurso de la partida. También se ofrece la posibilidad de salir de la aplicación escogiendo la opción específica del menú principal. Y por último, iniciar la partida al videojuego que nos conduciría de nuevo al entorno

anteriormente descrito, previa visualización durante la carga del escenario de un pequeño tutorial para saber cómo desenvolverse en el juego.

Como se ha citado con anterioridad, *Cube´s War* es la unión de dos proyectos. En uno se centra en la creación del entorno y de los elementos 3D necesarios más su dibujado en pantalla, y el restante, que corresponde a éste proyecto, produce y trata la lógica del videojuego en el que la interfaz creada por el anterior, originará su visualización.

La lógica del videojuego implica el uso y manipulación de cantidad de datos, como son la posición del personaje, de los enemigos, la colocación de los elementos que forman el escenario, los grados que la cámara debe girar, el salvado de las configuraciones…Cada uno de estos componentes tiene asociada su propia clase *Java* para almacenar sus variables y realizar las distintas operaciones sobre ellas.

Perteneciente igualmente al presente proyecto, se manipulan los distintos eventos capturados que se ocasionan sobre la pantalla táctil del dispositivo móvil cuando se está ejecutando una partida, con el fin de producir una determinada respuesta sobre el personaje que controla el usuario. Del mismo modo, se manejan los distintos valores que devuelve el sensor asociado a la aplicación, para producir sobre el entorno 3D del videojuego una rotación en el mismo sentido en el que el usuario gire el dispositivo móvil.

También en este proyecto se trata la reproducción de efectos de sonido ante distintas acciones del juego y/o la vibración del dispositivo móvil durante el transcurso de una partida.

Asimismo produce la presentación del menú principal del videojuego así como de las distintas opciones que se ofrecen al usuario para que realice las acciones asociadas y/o modificaciones sobre el videojuego. Estas configuraciones serán posteriormente guardadas cuando el usuario finalice la aplicación, para ser cargadas e implantadas al iniciar de nuevo el videojuego y no tener que establecerlas cada vez que lo inicie.

El manejo de los distintos estados de la aplicación, el correcto funcionamiento de la totalidad del videojuego y de todos sus componentes de tal modo que en el usuario produzca una experiencia notable, es parte de lo que en el presente proyecto se ha conseguido.

Capítulo 4

Desarrollo de la lógica de *Cube's War*

Este capítulo comprende las tres grandes fases que supone el desarrollo de un proyecto. El primer apartado, *4.1 Análisis*, contiene el Diagrama de flujo de la aplicación, los casos de uso, el diagrama de actividad y la captura de requisitos tanto de usuario como los de software. El segundo apartado, *4.2 Diseño conceptual*, presenta el diagrama de los módulos por los que está compuesta la aplicación y su posterior explicación. Y finalmente el último apartado, *4.3 Implementación*, se explican los detalles importantes para el desarrollo de la aplicación como las clases de las que está compuesta la lógica, el método para la detección de colisiones… justificando cada decisión tomada para obtener esos resultados.

4.1 Análisis

En este apartado se realiza un análisis de la aplicación desarrollada, en el que se incluye el análisis de los casos de uso y el diagrama de actividad del sistema. Finalmente se concluye con los requisitos de usuario, seguidos de los requisitos de software divididos en funcionales y no funcionales.

4.1.1 Diagrama de flujo

Los diagramas de flujo describen gráficamente la secuencia de los distintos pasos o etapas a seguir de un proceso, y su interacción. La representación gráfica favorece la comprensión del proceso, en este caso de la lógica del videojuego desarrollado, además de poder encontrar posibles mejoras. Seguidamente, se muestra el diagrama de flujo de la aplicación.

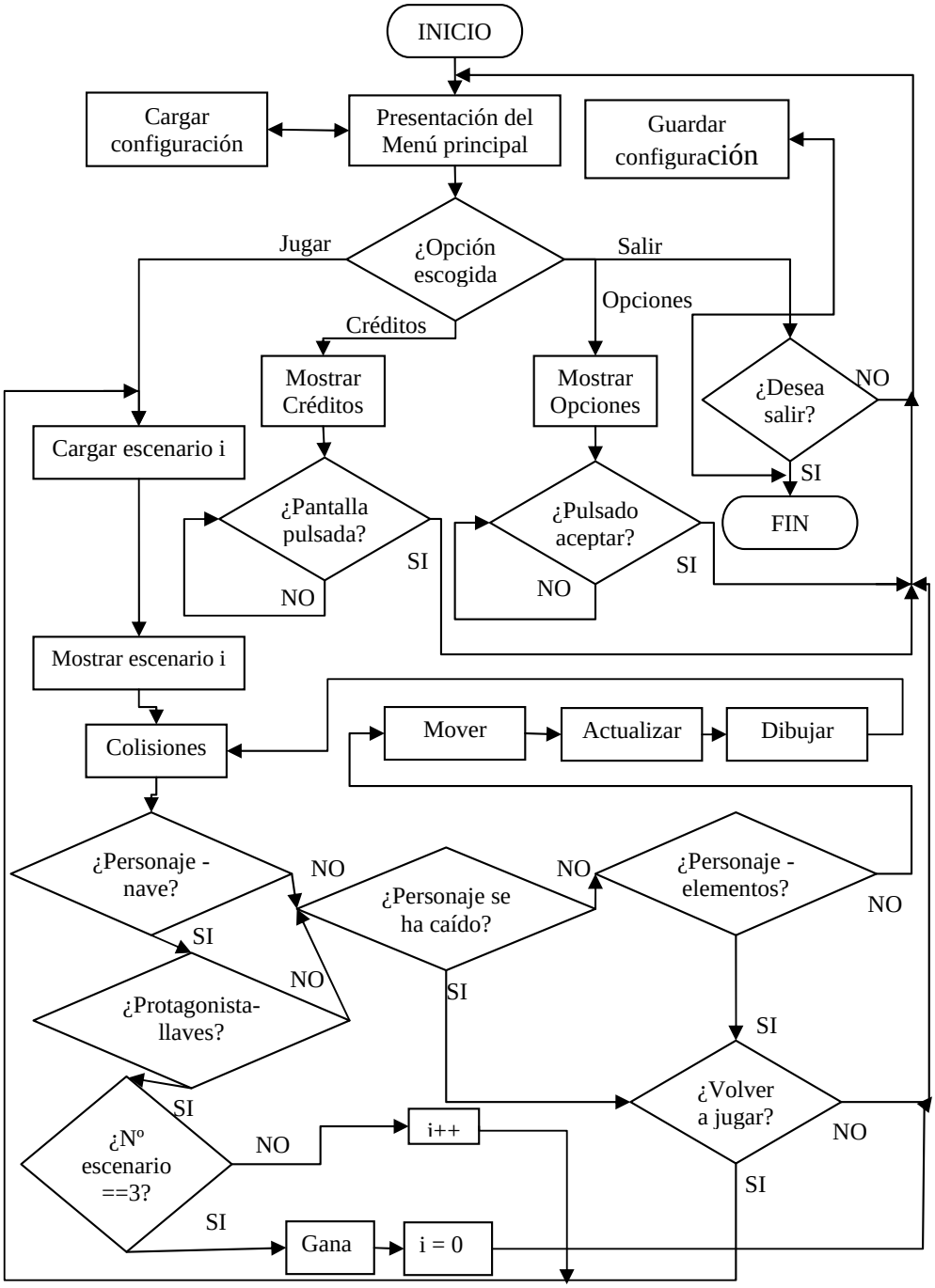


Figura 44. Diagrama de flujo de la aplicación

4.1.2 Casos de uso

Mediante el modelado de casos de uso se consigue identificar las funcionalidades del videojuego. Los casos de uso representan un uso típico del sistema, por lo que permite definir los requisitos que se deben cumplir en una aplicación y las interacciones que existen entre el usuario y el sistema. Se representan los casos de uso que resulten más claros al usuario con el fin de una mejor comprensión de la aplicación desarrollada.

Para el videojuego en el que nos centramos, se identifica un único actor que es el jugador o usuario de la aplicación.

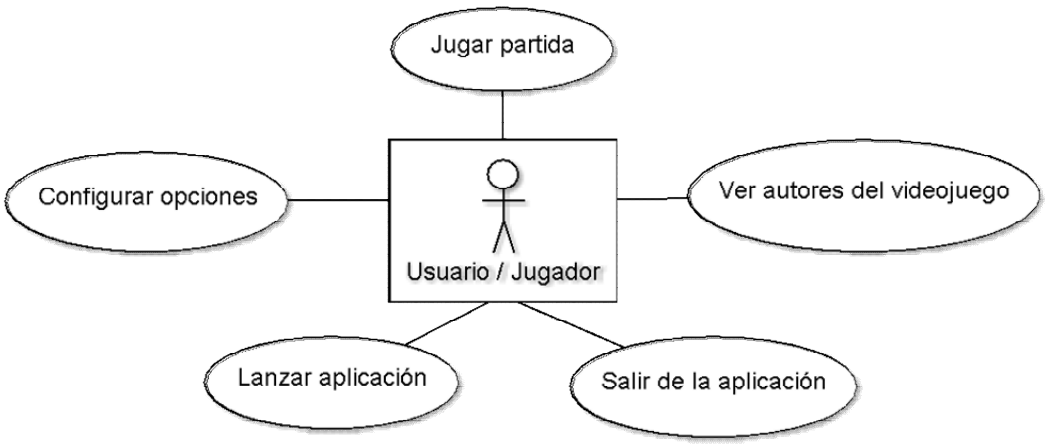


Figura 45. Casos de uso de la aplicación

Una vez definidos los casos de uso, cada uno de ellos se especifican siguiendo un formato tabular conteniendo la siguiente información para cada uno de ellos:

- **Identificador:** registra de forma unívoca un caso de uso siguiendo el formato de nombrado CU_X. Las X son valores numéricos secuenciales comprendidos entre 0 y 9.
- **Actor:** determina qué actor ha realizado dicho uso.
- **Descripción:** describe los pasos realizados por el usuario para la situación planteada.
- **Pre-condiciones:** condiciones que deben darse para la realización del caso de uso.
- **Post-condiciones:** condiciones que son resultado de la ejecución del caso de uso.

Identificador: Nombre	CU_1: Configurar opciones
Actor	Jugador
Descripción	1. El jugador visualiza el icono de la aplicación en su dispositivo móvil. 2. El jugador selecciona el icono. 3. El jugador visualiza el menú principal. 4. El jugador selecciona la opción "*Opciones*". 5. El jugador configura el sonido y la vibración.
Pre-condiciones	Situarse en el menú principal.
Post-condiciones	Mostrar menú principal.

Tabla 1: Caso de uso CU_1

Identificador: Nombre	CU_2:Ver autores del videojuego
Actor	Jugador
Descripción	1. El jugador visualiza el icono de la aplicación en su dispositivo móvil. 2. El jugador selecciona el icono. 3. El jugador visualiza el menú principal. 4. El jugador selecciona la opción "*Créditos*". 5. El jugador visualiza los autores.
Pre-condiciones	Situarse en el menú principal.
Post-condiciones	Mostrar menú principal.

Tabla 2: Caso de uso CU_2

Identificador: Nombre	CU_03: Salir de la aplicación
Actor	Jugador
Descripción	1. El jugador visualiza el icono de la aplicación en su dispositivo móvil. 2. El jugador selecciona el icono. 3. El jugador visualiza el menú principal. 4. El jugador selecciona la opción "*Salir*".
Pre-condiciones	Encontrarse en el menú principal.
Post-condiciones	Se cierra la aplicación.

Tabla 3: Caso de uso CU_3

Identificador: Nombre	CU_4: Lanzar aplicación
Actor	Jugador
Descripción	1. El jugador visualiza el icono de la aplicación en su dispositivo móvil. 2. El jugador selecciona el icono. 3. El jugador visualiza el menú principal.
Pre-condiciones	Seleccionar la aplicación en el dispositivo móvil.
Post-condiciones	Se visualiza el menú principal.

Tabla 4: Caso de uso CU_4

Identificador: Nombre	CU_5: Jugar partida
Actor	Jugador
Descripción	1. El jugador se encuentra en el menú principal. 2. El jugador selecciona la opción "*Jugar*". 3. Se muestra una pantalla de carga con información necesaria para saber jugar. 4. Se inicia la partida.
Pre-condiciones	Encontrarse en el menú principal.
Post-condiciones	• El jugador mueve al protagonista. • El jugador haga que salte el personaje. • El jugador mueve el dispositivo móvil para ver más del entorno. • El jugador pausa el juego. • El personaje colisiona con un elemento.

Tabla 5: Caso de uso CU_5

4.1.3 Diagrama de actividad

Un diagrama de actividad muestra los distintos estados de un sistema describiendo las secuencias que sigue éste. Estas transiciones influyen en el comportamiento y evolución del sistema provocando que éste cambie de un estado a otro como consecuencia de eventos que se producen.

Los casos de uso nos mostraban las distintas funcionalidades que tiene el sistema y con el diagrama de actividad se define el comportamiento del sistema ante los eventos que suceden.

A continuación se muestra el diagrama de actividad del sistema.

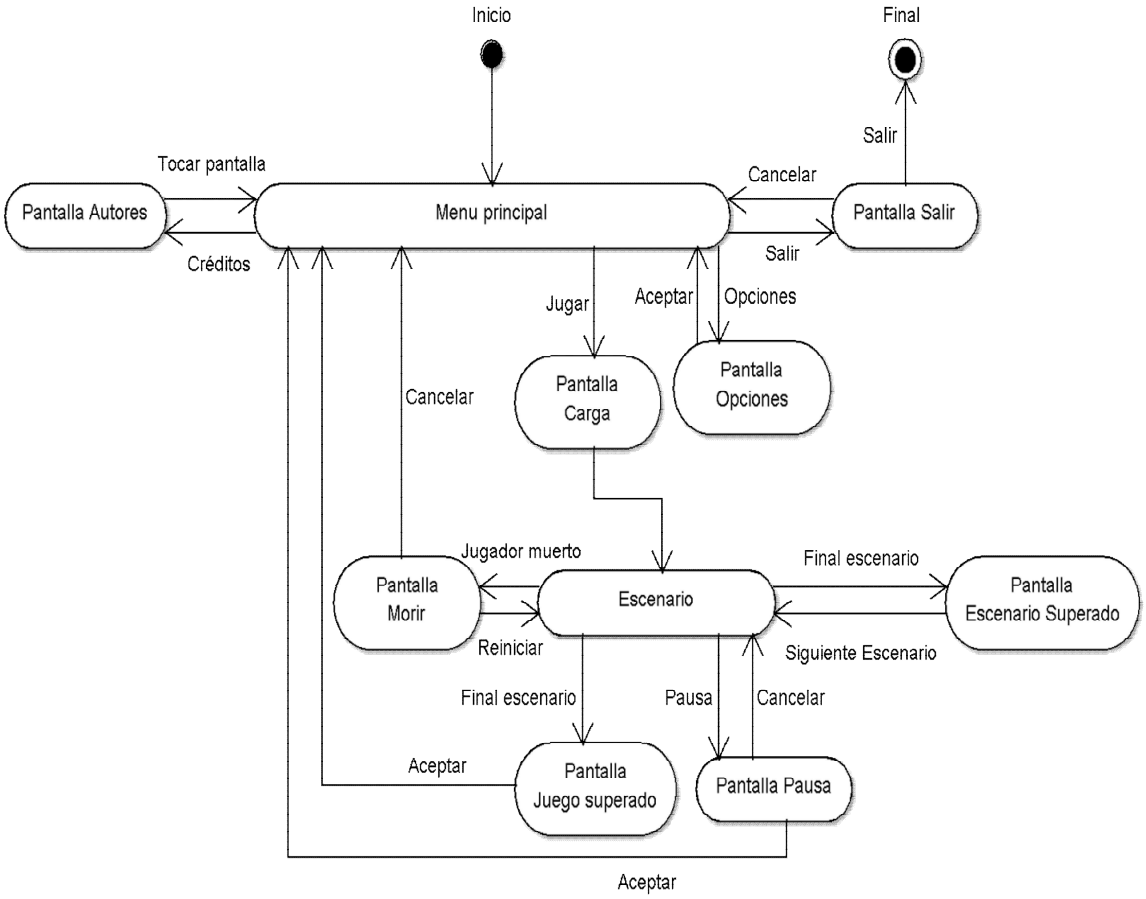


Figura 46. Diagrama de actividad del sistema

4.1.4 Requisitos de usuario

En este apartado del documento se identifican, clasifican y catalogan los distintos requisitos de usuario. Estos son especificaciones de las funcionalidades del sistema que han de ser resueltas. Cada requisito está especificado siguiendo un formato tabular conteniendo la siguiente información para cada uno de ellos:

- **Identificador:** registra de forma unívoca un requisito, asignando a cada uno el nombre RUV_X para los requisitos del videojuego y con el nombre de RUM_X a los del proyecto que he desarrollado. El valor de X es un identificador numérico y secuencial de los requisitos de cada categoría.
- **Descripción:** especificación detallada del requisito.
- **Origen o fuente:** el caso de uso del que proviene.
- **Verificabilidad:** define si el requisito se puede acreditar mediante una prueba.
- **Claridad:** determina si el requisito se entiende con precisión, o por el contrario, si su definición es ambigua.
- **Prioridad:** establece la importancia del requisito dentro del proyecto. Toma tres valores: "Alta", "Media" o "Baja".
- **Necesidad:** establece la obligación del cumplimiento del requisito dentro del proyecto. Toma dos valores: "Esencial" u "Opcional".
- **Estabilidad:** establece el grado de variabilidad que puede tener un requisito a lo largo del proyecto. Toma tres valores: "Alta", "Media" o "Baja".

4.1.4.1 Requisitos de usuario del videojuego

A continuación, se describen y especifican cada uno de los requisitos de usuario centrado en el videojuego desarrollado:

Identificador	RUV_1: Sistema *Android*
Descripción	El dispositivo móvil del usuario final debe contener el sistema operativo *Android* con versión igual o superior a la 2.2 (Froyo).
Origen o Fuente	Lanzar aplicación
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 6: Requisito de usuario RUV_1

Identificador	RUV_2: Almacenamiento
Descripción	Se requiere un espacio mínimo de almacenamiento interno para la instalación de la aplicación de 1,20MB en el dispositivo móvil.
Origen o Fuente	Lanzar aplicación
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 7: Requisito de usuario RUV_2

Identificador	RUV_3: Permisos
Descripción	Durante la instalación del juego, el usuario debe otorgar los permisos necesarios a la aplicación para que ésta pueda funcionar de forma correcta.
Origen o Fuente	Lanzar aplicación
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Baja

Tabla 8: Requisito de usuario RUV_3

Identificador	RUV_4: Tamaño visualización
Descripción	El tamaño de la aplicación debe ser optimizado para cualquier dimensión existente de pantalla.
Origen o Fuente	Lanzar aplicación
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Baja

Tabla 9: Requisito de usuario RUV_4

Identificador	RUV_5: Apariencia
Descripción	El videojuego debe presentar una interfaz clara y precisa para visualizar opciones, personajes, escenarios y elementos de éstos.
Origen o Fuente	Lanzar aplicación
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Baja

Tabla 10: Requisito de usuario RUV_5

Identificador	RUV_6: Perspectiva
Descripción	El usuario mediante el movimiento del dispositivo móvil, debe poder visualizar una mayor perspectiva de los escenarios.
Origen o Fuente	Jugar partida
Verificabilidad	SI
Claridad	SI
Prioridad	Media
Necesidad	Esencial
Estabilidad	Baja

Tabla 11: Requisito de usuario RUV_6

Identificador	RUV_7: Objetivo
Descripción	El videojuego debe de tener un objetivo claro que es el de tratar de sortear los obstáculos de los escenarios avanzando en ellos, esquivando a los enemigos y recogiendo la llave necesaria para poder superar el nivel.
Origen o Fuente	Jugar partida
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Opcional
Estabilidad	Alta

Tabla 12: Requisito de usuario RUV_7

4.1.4.2 Requisitos de usuario del proyecto

Se presenta, seguidamente, los requisitos de usuario para el módulo lógica del videojuego que he desarrollado:

Identificador	RUM_1: Menú Principal
Descripción	La aplicación debe presentar un menú principal en el que el usuario pueda seleccionar las opciones que se presenten.
Origen o Fuente	Lanzar aplicación
Verificabilidad	SI
Claridad	SI
Prioridad	Media
Necesidad	Opcional
Estabilidad	Baja

Tabla 13: Requisito de usuario RUM_1

Identificador	RUM_2: Opciones
Descripción	Se debe proporcionar al usuario la posibilidad de configurar opciones del juego como la reproducción de efectos de sonido y la vibración del dispositivo móvil.
Origen o Fuente	Configurar opciones
Verificabilidad	SI
Claridad	SI
Prioridad	Media
Necesidad	Opcional
Estabilidad	Baja

Tabla 14: Requisito de usuario RUM_2

Identificador	RUM_3: Tutorial del juego
Descripción	Debe existir una pequeña explicación de cómo se juega al videojuego.
Origen o Fuente	Jugar partida
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Opcional
Estabilidad	Alta

Tabla 15: Requisito de usuario RUM_3

Identificador	RUM_4: Salir
Descripción	El usuario debe poder salir de la aplicación en cualquier momento.
Origen o Fuente	Salir de la aplicación
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 16: Requisito de usuario RUM_4

Identificador	RUM_5: Créditos
Descripción	El usuario debe poder conocer la identidad de los autores del videojuego.
Origen o Fuente	Ver autores del videojuego
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 17: Requisito de usuario RUM_5

Identificador	RUM_6: Control personaje
Descripción	El usuario controla el movimiento del personaje protagonista.
Origen o Fuente	Jugar partida
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 18: Requisito de usuario RUM_6

Identificador	RUM_7: Información
Descripción	Se deben mostrar distintas pantallas informando de los eventos que se pueden producir como que se ha superado el escenario o la totalidad del juego, que ha sido eliminado…entre otros.
Origen o Fuente	Jugar partida
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Opcional
Estabilidad	Alta

Tabla 19: Requisito de usuario RUM_7

Identificador	RUM_8: Lógica eficaz
Descripción	La lógica del juego desarrollado debe ser eficiente en recursos y ejecución, de tal manera que no se produzca retardos durante la actuación de ésta.
Origen o Fuente	Jugar partida
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Opcional
Estabilidad	Alta

Tabla 20: Requisito de usuario RUM_8

Identificador	RUM_9: Colisiones
Descripción	Debe de existir un buen control de colisiones entre el personaje y los enemigos o elementos del escenario, y entre ellos.
Origen o Fuente	Jugar partida
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Opcional
Estabilidad	Alta

Tabla 21: Requisito de usuario RUM_9

4.1.5 Requisitos de software

En este apartado se identifican, clasifican y catalogan los principales requisitos software del sistema. Estos proporcionan una visión completa del comportamiento global del sistema. Cada requisito está descrito siguiendo un formato tabular conteniendo la siguiente información para cada uno de ellos:

- **Identificador:** registra de forma unívoca un requisito, asignando a cada uno el nombre RSF_XX para los requisitos funcionales, y con RSNF_XX para los que no lo son. Las X son valores numéricos secuenciales comprendidos entre 00 y 99.

- **Descripción:** especificación detallada del requisito.

- **Origen o fuente:** el requisito de usuario del que proviene.

- **Verificabilidad:** define si el requisito se puede acreditar mediante una prueba.

- **Claridad:** determina si el requisito se entiende con precisión, o por el contrario, si su definición es ambigua.

- **Prioridad:** establece la importancia del requisito dentro del proyecto. Toma tres valores: "Alta", "Media" o "Baja".

- **Necesidad:** establece la obligación del cumplimiento del requisito dentro del proyecto. Toma dos valores: "Esencial" u "Opcional".

- **Estabilidad:** establece el grado de variabilidad que puede tener un requisito a lo largo del proyecto. Toma tres valores: "Alta", "Media" o "Baja".

Se debe destacar que dentro de los requisitos no funcionales existe otra clasificación:

- **Interfaz:** son aquellos requisitos que hacen referencia a la interfaz del sistema a través de la cual el usuario deberá interactuar.

- **Documentación:** aquellos requisitos que estén relacionados con temas de ayudas y tutoriales.

- **Operacionales:** son aquellos que hacen referencia al entorno de ejecución donde debe funcionar el sistema.

- **Rendimiento:** requisitos que garantizan el buen funcionamiento de la aplicación.

- **Seguridad ante Fallos:** requisitos que permiten a la aplicación recuperarse ante fallos u operaciones indebidas realizadas por los usuarios.

- **Recursos:** aquellos que la aplicación necesita para su correcto funcionamiento.

4.1.5.1 Requisitos funcionales

Identificador	RSF_01: Detectar eventos sensor
Descripción	El videojuego debe capturar los eventos del sensor de orientación del dispositivo móvil para mediante el movimiento de éste poder visualizar más perspectiva del escenario.
Origen o Fuente	Lógica eficaz
Verificabilidad	SI
Claridad	NO
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Media

Tabla 22: Requisito de software RSF_01

Identificador	RSF_02: Detectar eventos pantalla
Descripción	El videojuego debe capturar los distintos eventos sobre la pantalla táctil del dispositivo móvil para poder realizar distintas acciones.
Origen o Fuente	Lógica eficaz
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 23: Requisito de software RSF_02

Identificador	RSF_03: Mostrar menú principal
Descripción	Al iniciar la aplicación se debe mostrar el menú principal.
Origen o Fuente	Menú principal
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 24: Requisito de software RSF_03

Identificador	RSF_04: Mostrar opciones
Descripción	Se debe mostrar las opciones del videojuego en un sub-menú y poder configurarlas.
Origen o Fuente	Opciones
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 25: Requisito de software RSF_04

Identificador	RSF_05: Mostrar autores
Descripción	Se debe mostrar los autores del videojuego en un sub-menú.
Origen o Fuente	Créditos
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 26: Requisito de software RSF_05

Identificador	RSF_06: Guardar configuraciones
Descripción	Al salir de la aplicación se debe guardar la configuración establecida por el usuario.
Origen o Fuente	Salir
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 27: Requisito de software RSF_06

Identificador	RSF_07: Mostrar pantallas informativas
Descripción	Si se elimina o supera el escenario, se debe de informar al usuario de tal evento.
Origen o Fuente	Información
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 28: Requisito de software RSF_07

4.1.5.2 Requisitos no funcionales

- **Requisitos de interfaz:**

Identificador	RSNF_01: Interfaz intuitiva
Descripción	La interfaz se debe presentar de manera clara al usuario para facilitar su uso.
Origen o Fuente	Apariencia
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Media

Tabla 29: Requisito de software RSNF_01

Identificador	RSNF_02: Interfaz legible
Descripción	La interfaz debe presentar la información de forma legible.
Origen o Fuente	Apariencia
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Media

Tabla 30: Requisito de software RSNF_02

Identificador	RSNF_03: Interfaz interactiva
Descripción	La interfaz debe requerir de la interacción del usuario.
Origen o Fuente	Apariencia
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 31: Requisito de software RSNF_03

- **Requisitos de documentación:**

Identificador	RSNF_04: Cómo jugar
Descripción	Debe existir una pequeña explicación de cómo se juega durante la carga de los escenarios.
Origen o Fuente	Tutorial del juego
Verificabilidad	SI
Claridad	SI
Prioridad	Media
Necesidad	Opcional
Estabilidad	Baja

Tabla 32: Requisito de software RSNF_04

Identificador	RSNF_05: Autores del juego
Descripción	Debe haber información con la identidad de los autores del videojuego.
Origen o Fuente	Información
Verificabilidad	SI
Claridad	SI
Prioridad	Media
Necesidad	Opcional
Estabilidad	Baja

Tabla 33: Requisito de software RSNF_05

Identificador	RSNF_06: Datos
Descripción	Deben existir pantallas informativas para el usuario.
Origen o Fuente	Información
Verificabilidad	SI
Claridad	SI
Prioridad	Media
Necesidad	Opcional
Estabilidad	Baja

Tabla 34: Requisito de software RSNF_06

- **Requisitos operacionales:**

Identificador	RSNF_07: Idioma
Descripción	El idioma del videojuego en el que se debe presentar es el español.
Origen o Fuente	Apariencia
Verificabilidad	SI
Claridad	SI
Prioridad	Baja
Necesidad	Opcional
Estabilidad	Baja

Tabla 35: Requisito de software RSNF_07

Identificador	RSNF_08: Salir
Descripción	Se debe poder salir de la aplicación.
Origen o Fuente	Salir
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 36: Requisito de software RSNF_08

Identificador	RSNF_09: Operaciones
Descripción	El número de operaciones que se debe realizar en el bucle principal del juego tiene que ser eficaces.
Origen o Fuente	Lógica eficaz
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 37: Requisito de software RSNF_09

Identificador	RSNF_10: Detección de colisiones
Descripción	La detección de colisiones entre los distintos elementos debe ser inmediata y realista.
Origen o Fuente	Colisiones
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 38: Requisito de software RSNF_10

- **Requisitos de rendimiento:**

Identificador	RSNF_11: Tiempo de carga Menús
Descripción	El tiempo de carga del menú principal y de cualquiera de sus sub-menús debe ser inferior a 1 segundo.
Origen o Fuente	Menú principal
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 39: Requisito de software RSNF_11

Identificador	RSNF_12: Tiempo de carga de escenarios
Descripción	El tiempo de inicialización y carga de los escenarios al comenzar una partida debe ser inferior a 3 segundos.
Origen o Fuente	Lógica eficaz
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 40: Requisito de software RSNF_12

Identificador	RSNF_13: Tiempo Pulsación
Descripción	El tiempo de detección de pulsación de la pantalla mínimo.
Origen o Fuente	Control personaje
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 41: Requisito de software RSNF_13

Identificador	RSNF_14: Eficiencia
Descripción	Se deben aprovechar los recursos disponibles para una veloz ejecución de la aplicación.
Origen o Fuente	Lógica eficaz
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 42: Requisito de software RSNF_14

- **Requisitos de seguridad frente a errores:**

Identificador	RSNF_15: Control errores
Descripción	El sistema debe ofrecer opciones cerradas y un control del entorno gráfico para evitar posibles errores.
Origen o Fuente	Lógica eficaz
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 43: Requisito de software RSNF_15

- **Requisitos de recursos:**

Identificador	RSNF_16: Liberación de recursos
Descripción	Los recursos vinculados a la aplicación, cuando no se necesaria su actuación, deben ser liberados.
Origen o Fuente	Lógica eficaz
Verificabilidad	SI
Claridad	SI
Prioridad	Alta
Necesidad	Esencial
Estabilidad	Alta

Tabla 44: Requisito de software RSNF_16

Identificador	RSNF_17: Ahorro batería
Descripción	Se deben utilizar sonidos con frecuencias de muestreo mínimas, imágenes con poca profundidad de bits y un mínimo consumo de recursos para que de esta manera no se produzca un gasto excesivo de batería.
Origen o Fuente	Lógica eficaz
Verificabilidad	SI
Claridad	SI
Prioridad	Media
Necesidad	Opcional
Estabilidad	Media

Tabla 45: Requisito de software RSNF_17

- **Requisitos de usabilidad:**

Identificador	RSNF_18: Facilidad de uso
Descripción	No se debe necesitar de una preparación previa para que el usuario juegue al mismo.
Origen o Fuente	Control personaje.
Verificabilidad	SI
Claridad	SI
Prioridad	Baja
Necesidad	Opcional
Estabilidad	Baja

Tabla 46: Requisito de software RSNF_18

Identificador	RSNF_19: Personas dirigidas
Descripción	El videojuego debe dirigirse a cualquier persona.
Origen o Fuente	Control personaje.
Verificabilidad	SI
Claridad	SI
Prioridad	Baja
Necesidad	Opcional
Estabilidad	Baja

Tabla 47: Requisito de software RSNF_19

4.2 Diseño conceptual

Se procede en este apartado a presentar el diagrama de los módulos en los que se divide la aplicación y, posteriormente, se realiza la explicación de cuál es la función de cada uno de ellos individualmente y sus posibles vínculos con los restantes módulos.

4.2.1 Arquitectura del sistema

El videojuego *Cube´s War* se divide en un conjunto de módulos relacionados entre sí, de modo que se minimicen las dependencias entre ellos y poder separarlos con facilidad. Cada uno tiene una funcionalidad específica y la unión de todos los módulos producen como resultado la aplicación desarrollada.

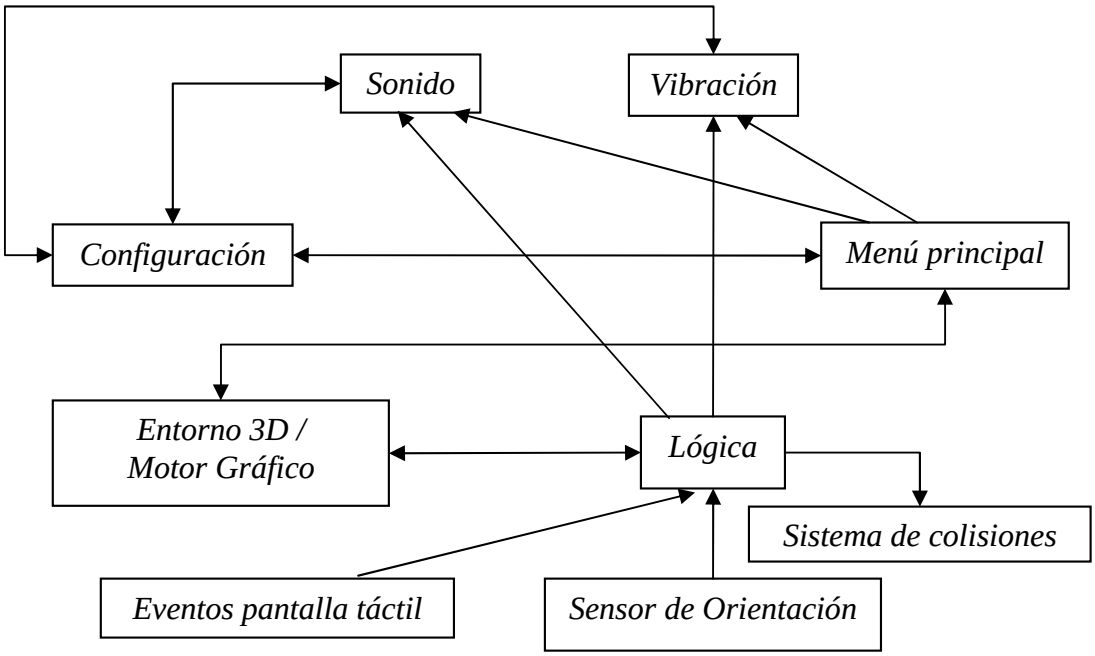


Figura 47. Arquitectura del sistema

4.2.2 Descripción de los módulos

Una vez observada las relaciones de dependencia existente entre los módulos, se dispone seguidamente a exponer el papel de cada uno de ellos en el conjunto y centrado en la visión global de su funcionamiento. Se realiza de forma más breve en aquellos módulos que no pertenezcan al presente proyecto, siendo solo en este caso el llamado *Entorno 3D/Motor Gráfico*.

- **Módulo *Menú principal*:** es el encargado de mostrar el menú principal cuando se inicia la aplicación y de realizar todas las acciones que se ofrecen al usuario, como modificar el estado de habilitación del módulo *Sonido* y *Vibración* o mostrar los autores que han desarrollado el proyecto. También se ofrece la posibilidad de salir de la aplicación, por lo que el módulo de *Configuración* entraría en ejecución. Además es a partir de *Menú principal* que origina el *Entorno 3D/Motor Gráfico*, que es el contexto donde se desarrolla toda la lógica del videojuego, volviendo a éste cuando se finalice la partida. Por lo tanto, es considerado el núcleo principal de todo el proyecto, dado que es a partir de él que se deriva hacia las distintas posibilidades que la aplicación ofrece, retornando a éste una vez finalizadas, pudiendo producir distintos eventos dependiendo de la causa.

- **Módulo *Sonido*:** se ocupa de la reproducción de distintos efectos de sonido dependiendo de las acciones que se produzcan durante el desarrollo del videojuego, tanto en el *Menú principal* como en *Lógica*. También, mientras que la aplicación se encuentre en ejecución, maneja el estado de habilitación de reproducción, para que posteriormente cuando se finalice, el módulo de *Configuración* guarde ese estado. Además será éste que implantará al módulo *Sonido* esa capacitación cuando se inicie de nuevo la aplicación. No es un

módulo clave para el correcto funcionamiento del sistema, dado que solo reproduce sonidos y no modifica o altera parámetro alguno que pueda intervenir en el desarrollo lógico del juego, conducir a errores o a estados no deseados.

- **Módulo *Vibración*:** su cometido es producir la vibración del dispositivo móvil una determinada duración ante distintas acciones de la aplicación, es decir, en aquellas que se ocasionen en el *Menú principal* y en *Lógica*. Al igual que en el módulo *Sonido*, manipula la condición que posibilita la vibración, que será almacenado por *Configuración* cuando se cierre la aplicación, y de nuevo implantado por éste cuando se vuelva a iniciar. Tampoco es un módulo esencial en la ejecución del videojuego, solo realiza su función para producir sensaciones al usuario conforme con la acción que ha sucedido.

- **Módulo *Configuración*:** es el encargado de guardar en un fichero de configuración el estado de habilitación tanto del módulo de *Sonido* como el de *Vibración* cuando se produce el cierre de la aplicación, es decir, el módulo *Menú principal*. También almacena en ese fichero el número de escenario en el que se encontraba el usuario en el momento que decidió salir de la aplicación. Cuando posteriormente decida volver a iniciarla, se cargará todo lo almacenado en la configuración para conseguir de este modo que la aplicación se encuentre en el mismo estado en el que lo dejó el usuario. Es un módulo relativamente importante porque su cometido produce sobre el usuario la sensación de personalización de la aplicación, pudiendo considerarse opcional.

- **Módulo *Entorno 3D / Motor Gráfico*:** este módulo no se implementa en este proyecto, sino que es una biblioteca desarrollada en otro PFC. Se ocupa de la preparación del entorno 3D donde se va a desarrollar el videojuego, su distribución, los elementos que puede usar el módulo *Lógica* para determinar cómo quiere que sea la estética de sus componentes, las imágenes *Sprites* de los componentes y el dibujado de ellos. Es la interfaz gráfica del módulo *Lógica*, y que sin éste, no se visualizaría elemento alguno sobre la pantalla, por lo que el usuario no podría disfrutar del videojuego. Consecuentemente, posee una elevada importancia y es indispensable dentro del conjunto de la aplicación.

- **Módulo *Lógica*:** sistema que controla la totalidad del videojuego. Se subdivide en pequeñas partes para que cada una de ellas se centren en el manejo de los distintos componentes que conforman el videojuego. Una de estas unidades es la del protagonista, que se encarga de recibir los datos del módulo *Eventos pantalla táctil* para producir el avance del personaje controlado, con la ayuda de *Sistema de colisiones*. Otro componente son los enemigos que pueden ser de tres tipos con distintas características, y que determinando la posición del protagonista anterior, producen su avance sobre el escenario determinado por *Sistema de Colisiones*. También están los meteoritos que son componentes que tienen un movimiento de caída sobre el escenario que el protagonista debe sortear. El control de la cámara es otra unidad, la cual recibe los datos capturados por el módulo de *Sensor de Orientación* para su tratamiento y establecimiento de límites, que producen sobre *Entorno 3D / Motor Gráfico* un giro en el sentido en el que el usuario rote el dispositivo móvil. Los escenarios son parte también del módulo *Lógica* y que en ellos se conforma cómo van a ser y qué determinados elementos 3D se necesitan para su dibujado. Y por último, la lógica del

videojuego en sí, en el que el bucle principal de éste lo utiliza para determinar las colisiones que se pueden producir entre los distintos elementos por el módulo *Sistema de colisiones*, producir el movimiento de las posiciones de los componentes descritos con anterioridad y finalmente la actualización de sus datos. Todas estas unidades descritas con anterioridad conllevan el manejo de cantidad de datos y conforman el desarrollo total del videojuego, por lo tanto, es un módulo fundamental e indispensable, considerándose como otro núcleo junto con el del *Menú principal*. Estos datos son usados por el módulo de *Entorno 3D / Motor Gráfico* para dibujar los elementos.

- **Módulo *Sistema de colisiones*:** parte vital dentro del videojuego y que lo utiliza el módulo *Lógica* para determinar si se puede seguir con el desarrollo del videojuego. Este módulo produce que el videojuego tenga un desarrollo lógico y que se traten distintas acciones que se pueden producir durante una partida. En él se determina si se ha producido colisión alguna entre: protagonista-escenario, protagonista-enemigos, protagonista-meteoritos, protagonista-pinchos, protagonista-llave, protagonista-puerta, enemigos-escenario, enemigos-pinchos y finalmente meteoritos-escenario.

- **Módulo *Eventos pantalla táctil*:** su misión es la de capturar los distintos eventos que se producen sobre la pantalla táctil del dispositivo móvil en el que se ejecute la aplicación. Entra en acción cuando se ha iniciado una partida, para determinar qué evento se ha producido, sobre qué pixeles concretos ha ocurrido la pulsación y si éstos se encuentran sobre la zona delimitada para los botones de movimiento del personaje que controla el usuario. Posteriormente a la realización de estas comprobaciones, actuará el módulo *Lógica* en función del resultado que se haya obtenido, produciendo el movimiento o no del personaje controlado. Su cometido es esencial e indispensable para el desarrollo normal del videojuego, dado que sin él, el usuario no podría hacer progresar al personaje a lo largo de los distintos escenarios que se presentan.

- **Módulo *Sensor de orientación*:** se ocupa de la captura y envío de los datos que proporciona el sensor de orientación del dispositivo móvil asociado a la aplicación, al módulo *Lógica* donde será tratados. Solo se comienza a capturar esos datos que se facilitan, una vez iniciada la aplicación. Estos son los que una vez tratados por *Lógica*, provocarán sobre el *Entorno 3D/Motor Gráfico* una rotación en el mismo sentido en el que el usuario gire el dispositivo móvil. Con el giro del entorno se puede visualizar si sobre el escenario existen componentes en el eje Z y poder avanzar o no en el sentido deseado. El módulo tiene una gran importancia dado que al ser un videojuego de plataformas en 3D, se tiene que facilitar al usuario algún modo de poder visualizar el entorno donde se encuentra el personaje que controla.

4.3 Implementación

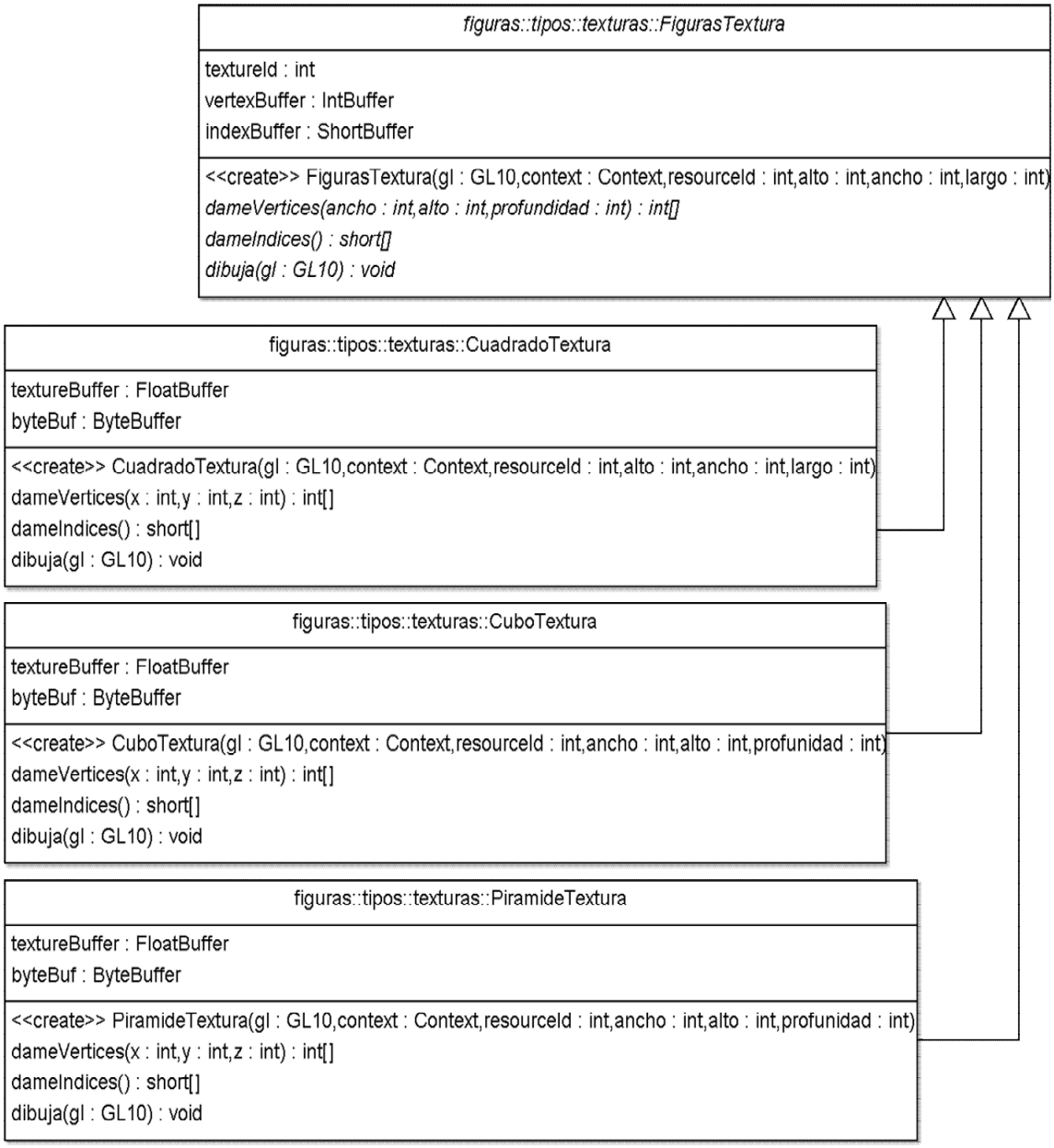
Una vez realizado el análisis y diseño de la aplicación, en este apartado se procede a su uso para llevar a cabo la implementación de la lógica del videojuego. No se comenta

todo el desarrollo de la aplicación, sino que se profundiza en los aspectos más importantes de su implementación, de este modo se detallan aspectos como la implementación del jugador así como su control y sus acciones, la detección de colisiones, los enemigos y su comportamiento, los escenarios, etc.

4.3.1 Contrato de interfaz

Seguidamente se muestran cada una de las clases de la biblioteca creada en otro PFC utilizada en este proyecto, organizadas por paquetes, incluyendo sus atributos y métodos, para de este modo conformar el *API* o interfaz desarrollada. Solo se muestran aquellas clases cuyos métodos son empleados por este proyecto.

- **Paquete *figuras.tipos.texturas*:**



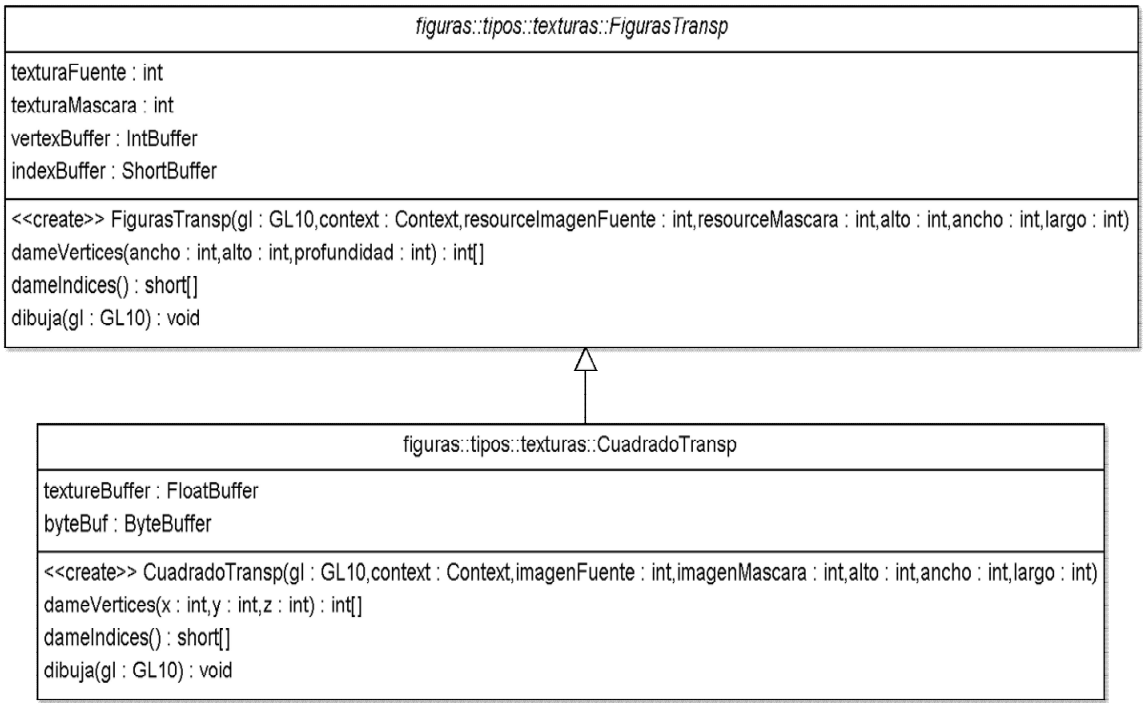


Figura 48. Clases del paquete *figura.tipos.texturas*

- **Paquete *dibujar.camara*:**

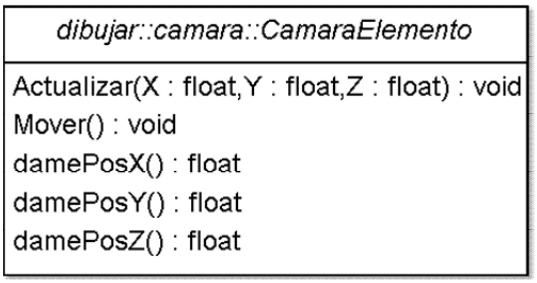
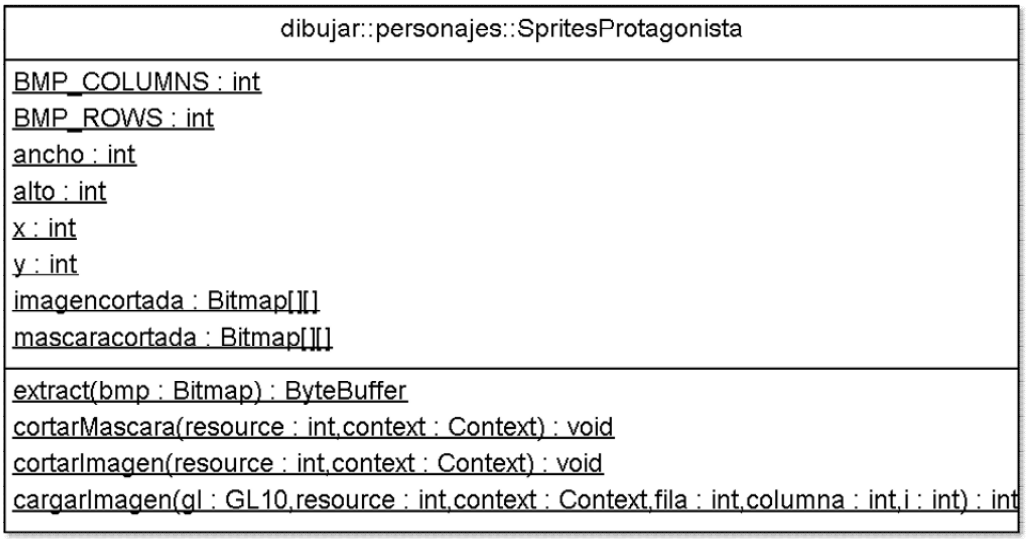


Figura 49. Clases del paquete *dibujar.camara*

- **Paquete *dibujar.personajes*:**



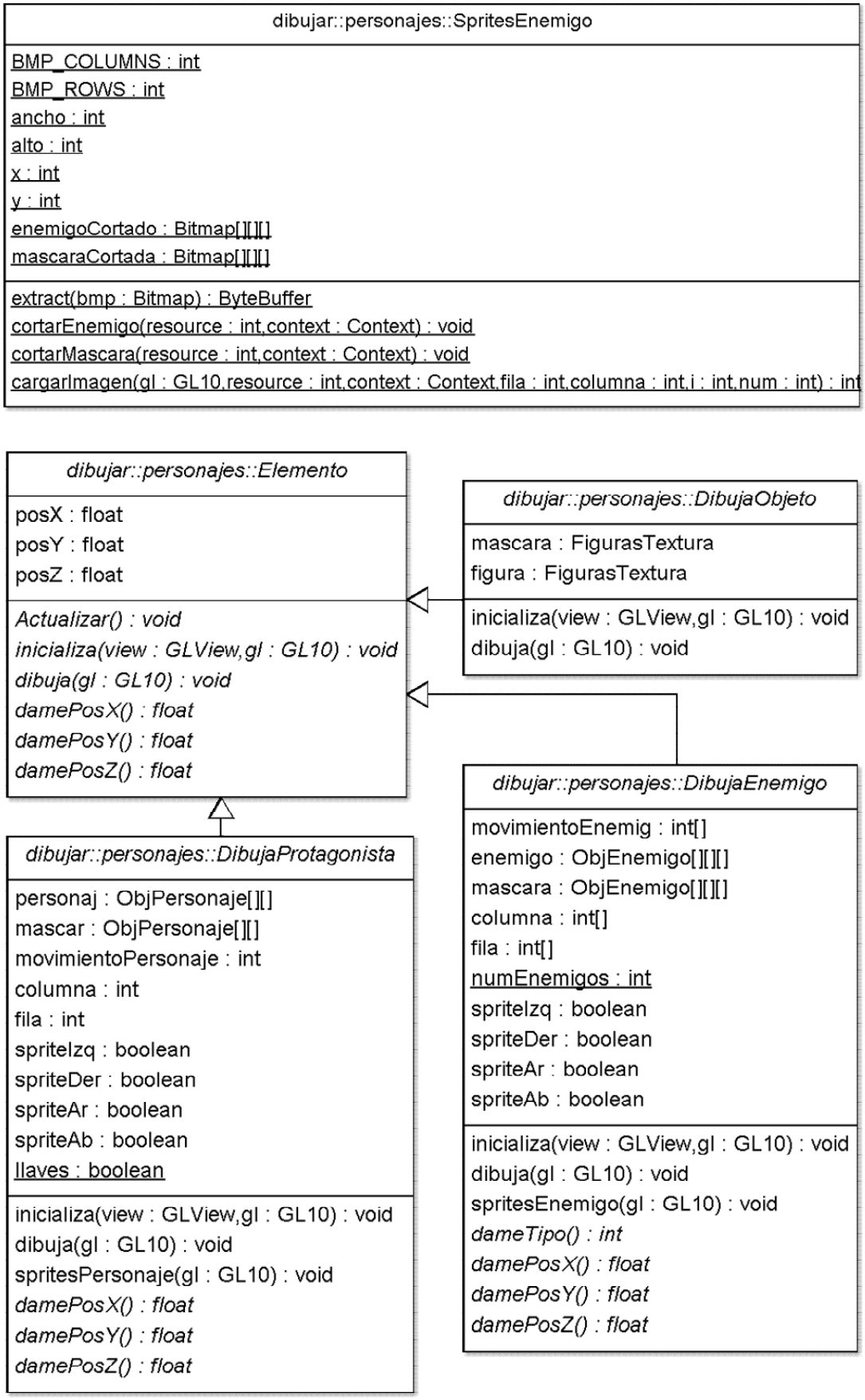


Figura 50. Clases del paquete *dibujar.personajes*

- **Paquete *dibujar.escenario*:**

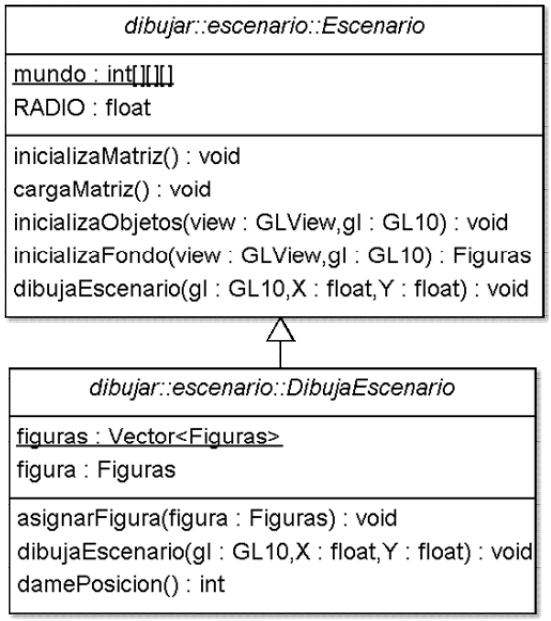


Figura 51. Clases del paquete *dibujar.escenario*

- **Paquete *proyecto.dibujar*:**

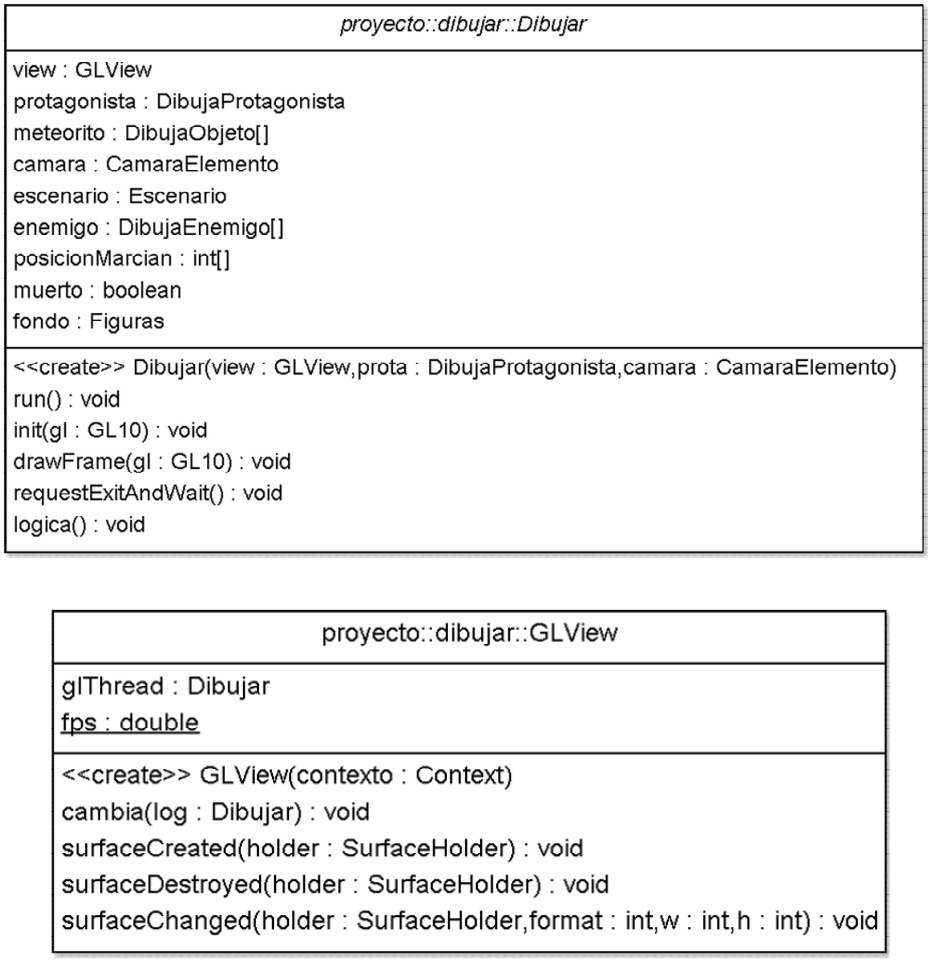


Figura 52. Clases del paquete *proyecto.dibujar*

4.3.2 Diagrama de clases

A continuación se muestran cada una de las clases desarrolladas en el presente proyecto, presentándose en un diagrama de clases como representación de todo el trabajo desarrollado, con el objetivo de definir las distintas relaciones que existen entre cada una de ellas para ofrecer una mayor claridad.

Más adelante se explica cada una de las clases mostradas y su función en el conjunto global del videojuego.

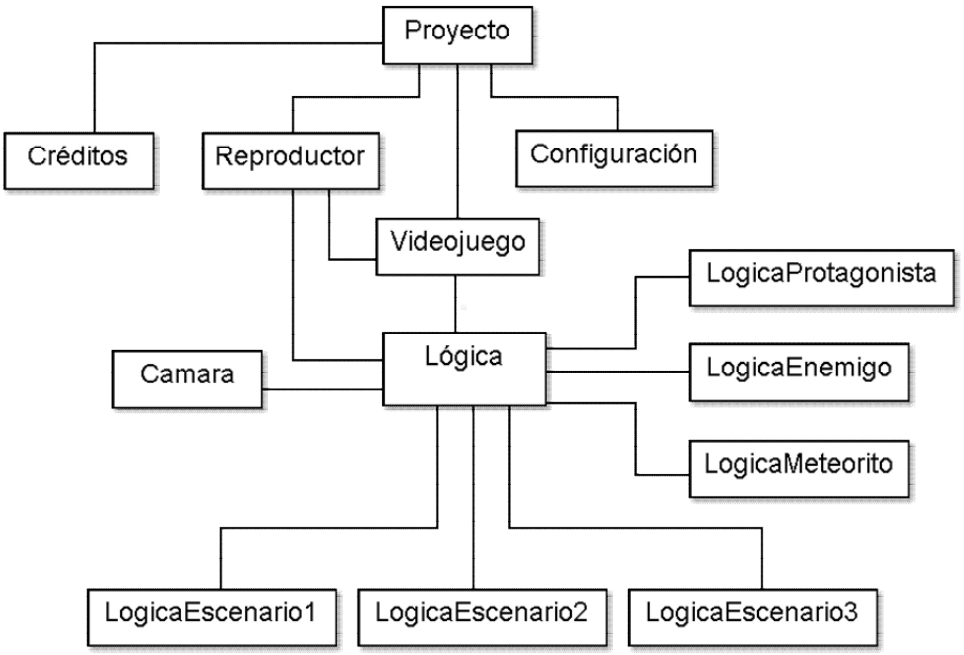


Figura 53. Diagrama de clases

4.3.3 Detalles de la implementación

Con este apartado de la memoria se pretende ahondar en los aspectos más importantes del desarrollo del videojuego dentro del módulo que me corresponde como es el de la lógica de éste. En él se describirán tanto la lógica del juego, como la implementación del jugador así como su control y sus acciones, la detección de colisiones, los enemigos y su comportamiento, la creación de los escenarios, sonidos, vibración etc.

El orden en el que se irán describiendo cada aspecto, será igual a como si se iniciara la aplicación para jugar, es decir, primero se visualizaría el menú principal y sus posibilidades, para posteriormente pasar al entorno donde se desenvuelve el juego con el escenario, personaje y enemigos.

Primero se debe recalcar que el juego desarrollado se despliega en un entorno de 3D. Esto se consigue gracias a que el S.O. *Android* incluye soporte para gráficos de alto rendimiento en 2D y 3D con *Open Graphics Library* (*OpenGL*), en concreto, la *API OpenGL ES*. *OpenGL* es una *API* gráfica multi-plataforma que especifica una interfaz de software estándar para el hardware de procesamiento de gráficos 3D.

OpenGL ES es una idea de la especificación *OpenGL* destinado a dispositivos embebidos. Sus versiones 1.0 y 1.1 de las especificaciones *API* han sido apoyados desde *Android* 1.0. Posteriormente, desde que *Google* sacara la versión 2.2 (Nivel API 8), la plataforma admite especificación *OpenGL ES* 2.0 *API*, siendo esta versión de la biblioteca compatible con la mayoría de los dispositivos *Android* y con la versión 1.1 de *OpenGL ES*[92].

Tal y como muestra los siguientes datos, el 90.6% de los dispositivos móviles activos que accedieron a *Android Market* durante los últimos 7 días del mes de Agosto de 2011[93], tienen soporte sobre la versión 2.0 de *OpenGL ES*, por lo que esto fue lo que hizo decantarse para desarrollar la aplicación sobre esta última, porque de este modo se abarcaría un mayor público.

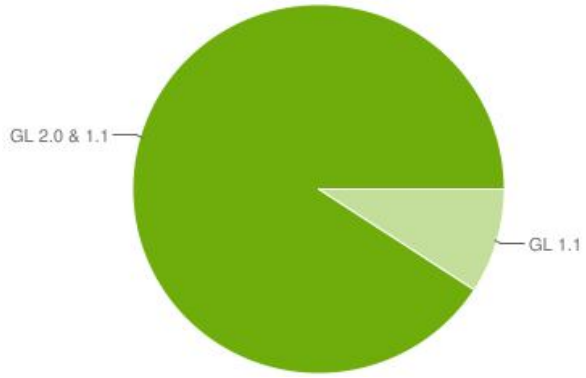


Figura 54. Gráfica de móviles con versiones OpenGL ES

Este requerimiento está especificado y declarado en el fichero *AndroidManifest.xml*[94], mediante el siguiente atributo:

```
<uses feature android:glEsVersion="0x00020000"/>
```

Código 1: Atributo *OPENGL ES*

La versión 2.0 de *OpenGL ES*, como se ha dicho con anterioridad, es soportada por dispositivos que tengan *Android* 2.2 o superior, correspondiente con el nivel 8 del *API*, por lo que en la aplicación también se detalla este requisito mediante el siguiente atributo en *AndroidManifest.xml*:

```
<uses sdk android:minSdkVersion="8"/>
```

Código 2: Atributo SDK

El fichero *AndroidManifest.xml* mencionado, es imprescindible en cualquier aplicación *Android*, por lo que a se dispone a su descripción dado que será en muchas ocasiones que se recurrirá a éste por distintos motivos. Éste, al erigirse un nuevo proyecto, el usuario no tiene que crearlo de la nada, si no que se genera automáticamente. Está escrito en *XML* y describe los componentes de los que consta la aplicación, conteniendo aspectos como las *Activities*[95] que los implementan, sus requisitos, los datos que pueden manejar o cuando deben ser ejecutado, los *Intents*[96], bibliotecas, el nombre de la aplicación... También en *AndroidManifest.xml* es donde se deben concretar los permisos que el desarrollador solicita utilizar para su aplicación, que pueden ser desde realizar llamadas de teléfono, controlar hardware del dispositivo, localización por *GPS*, acceso a Internet...

Igualmente se debe destacar otra característica de la aplicación, y es que está pensada para que se juegue con el dispositivo móvil en posición horizontal, es decir, girado 90º en el sentido opuesto a las agujas del reloj con respecto a su posición natural, para que aunque se voltee el dispositivo móvil, la disposición de los elementos en pantalla no cambie con la rotación.

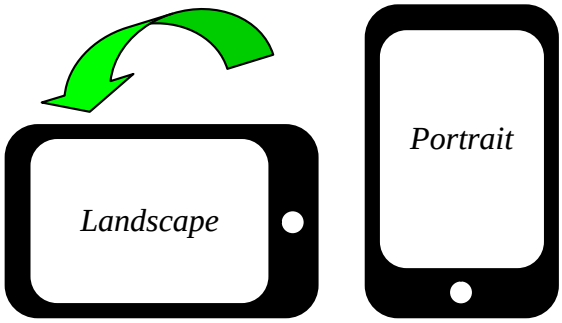


Figura 55. Tipos de orientación en dispositivos móviles

Esto se logra mediante la configuración del archivo *AndroidManifest.xml*, añadiendo a cada etiqueta de las *Activity* que conforman la aplicación, el atributo:

```
Android:screenOrientation="landscape"
```

Código 3: Atributo Orientación

Con esta modificación se obtiene la no rotación de los elementos, por lo que no hay que realizar una configuración por cada giro del dispositivo móvil. Pero esta característica no solo se ha realizado por reducir la dificultad, si no que además se ha tenido en cuenta que el juego desarrollado es del género plataformas. En estos juegos, el movimiento del personaje que se controla es esencial, por lo que si se determina una zona de la pantalla para posicionar un pad con el que se controlan sus movimientos, tiene que ser suficientemente ancha como para que no haya errores a la hora de determinar los botones que se han pulsado del pad. De este modo, determinando que la única orientación posible en la aplicación es la conocida como *landscape,* se obtiene una mayor zona de pulsación. Esto es debido a que los dispositivos móviles, por lo general, son más altos que anchos en dimensiones, pero si se cambia a esa orientación citada, se obtiene una anchura mayor. Asimismo, con este cambio se podrá visualizar una mayor zona del escenario cuando se inicie la partida.

Otro aspecto que se ha tenido muy en cuenta ha sido el ciclo de vida de una aplicación y el de la *Activity*. Una aplicación *Android* se ejecuta en su propio proceso. Este proceso es creado cuando parte del código necesita ser ejecutado, y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Cada componente de una aplicación tiene su propio ciclo de vida bien definido. Las *Activities* son uno de ellos, y a la hora de crear una aplicación, conviene tener en consideración su ciclo de vida, los distintos estados que puede tomar y la relación existente entre ellos propios.

En *Android*, cuando una nueva *Activity* es creada, las maneja como si de una pila se tratara, colocándola en lo más alto de ésta y relega la actividad anterior a permanecer justo debajo de ella en la propia pila. Esto provoca un cambio de estado en la propia

Activity, en el que *Android* permite al desarrollador poder controlar ese cambio y en cada momento saber en cuál se encuentra la *Activity*, consiguiendo así programar las acciones que mejor convengan y tomar decisiones adecuadas.

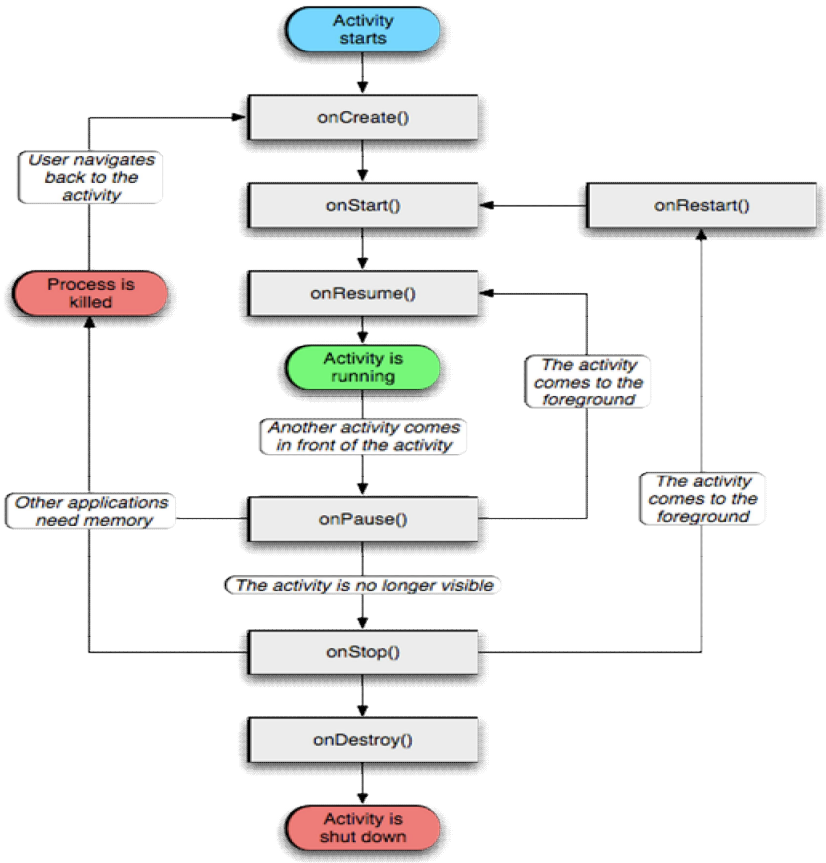


Figura 56. Ciclo de vida de un *Activity* [97]

El ciclo de vida de una *Activity* está profundamente relacionado con el de una aplicación, debido a que toda *Activity* tiene un ciclo de vida desde que se inicia la aplicación, hasta que se cierra, en el caso de una aplicación con una sola.

Cube´s War está compuesta por tres *Activities* denominadas *Proyecto, Videojuego y Créditos*, y en ellas se implementan cuatro estados de todos los que existen, por considerarse los más importantes:

- **onCreate():** se ejecuta cuando se crea la *Activity* por primera vez. Es aquí donde se deben crear *Views*, linkar datos a listas… en definitiva, el proceso de inicialización de la aplicación.

- **onPause():** cuando se tenga dos *Activities* en ejecución, la *Activity* que está en primer plano relega a la segunda *Activity* a un segundo plano actualizando esta al estado de pausa guardando su estado. Se utiliza para guardar datos que no se han grabado anteriormente o detener acciones que consuman CPU o batería.

- **onResume():** este método es llamado cuando una *Activity* que estaba en *onPause()*, vuelve al primer plano de ejecución, es decir, a su estado normal.

- **onDestroy():** es la última llamada antes de destruir la actividad. Puede darse porque la actividad está acabando o debido a que el sistema destruirá la instancia por necesidad.

Por ultimo, en esta aplicación se hace uso de las fuentes que *Android* pone a disposición. Estas fuentes pueden ser archivos *XML* de distribución de pantalla, o que contengan cadenas de texto y/o definiendo colores, animaciones, archivos de imagen y/o sonido, siendo un conjunto de recursos para uso de la aplicación. Todos están ubicados y distribuidos en sus respectivas carpetas dentro del directorio *res/*, y que cuando se crea algún archivo, o se modifican, se añaden ficheros… esto provoca que el plugin *ADT*[98] de *Android* sobre *Eclipse*, actualice una clase *Java* de la aplicación llamada *R.java* alojada en el directorio *gen/*, que contiene los IDs únicos de los elementos contenidos en esas carpetas o archivos *XML*. Cargando estos IDs permiten acceder al contenido de estos recursos mediante la llamada al método correspondiente, e incluyendo *R.id.xx*, o *R.string.xx, R.layout.xx, R.anim.xx, R.raw.xx, R.drawable.xx…* dependiendo de qué es lo que se quiera cargar en cada momento.

Una vez explicadas estas características de la aplicación desarrollada, se disponen a describir más detalles de la implementación del juego.

4.3.3.1 Reproducción de sonidos

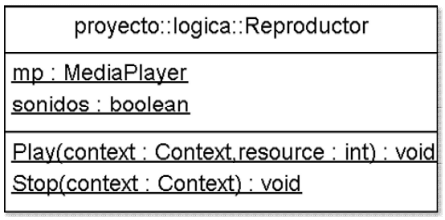


Figura 57. Clase *Reproductor*

El videojuego desarrollado reproduce efectos de sonido antes distintas acciones del usuario como: pulsar los botones, presionar botones físicos del dispositivo móvil, al conseguir completar un escenario o la totalidad del juego, si el usuario mediante el control de su personaje provoca que salte, choque con los enemigos, caiga de la plataforma por donde se desplaza o sobre los pinchos que se disponen por el escenario y finalmente colisione con los meteoritos que caen de la parte superior del escenario.

Para este cometido se ha creado una clase dedicado a ello llamada *Reproductor*. Ésta contiene un *boolean* que indica si se permite la reproducción de sonidos y se modifica mediante la configuración de las opciones del juego dentro del menú principal o cuando se cargan las configuraciones al iniciar el juego.

La clase *Reproductor* además del *boolean* anterior, contiene dos métodos que realizan el cometido al que está dedicado:

- **Play (Context context, int resource):** se utiliza para la reproducción de sonidos. Se tienen que facilitar en su llamada, el *Context* de la *Activity* y el fichero de sonido almacenados en *res/raw* mediante la ID del fichero *R.java*, que se quiere reproducir en los momentos que se han establecido. Dentro de dicho método, se realiza la comprobación de si está habilitada la reproducción de

sonidos y si es así, se inicializa un reproductor, le pasa la fuente, inicia la reproducción y cuando finalice ésta se procederá a la liberación de recursos.

- **Stop(Context context):** la finalidad de este método es la de detener cualquier reproducción de sonido que se esté ejecutando en el momento en el que se precise, liberando los recursos que tenia asociada esa reproducción que estaba en curso.

Una vez implementado, mediante la realización de llamadas a estos métodos estáticos de la clase *Reproductor*, se origina la reproducción de los sonidos que se han establecido. Estos archivos de sonido tienen que ser de un determinado formato para que el dispositivo móvil pueda reproducirlos, siendo los siguientes los soportados por *Android*[99].

Se escoge el formato *.wav* de 8 y 16 bits *PCM lineal*, siendo éste un formato de calidad y a su vez de reducido peso.

4.3.3.2 Vibración

Cuando se ha hablado anteriormente de *AndroidManifest.xml*, se ha comentado que es en él donde se deben concretar los permisos que el desarrollador solicita utilizar para su aplicación. Esto es debido porque para las aplicaciones está prohibido por el sistema ejercer acciones restringidas como realizar llamadas de teléfono, leer y escribir la información de contactos o *sms*, disponer de acceso a *Internet* o utilizar un elemento hardware del dispositivo. La única manera de tener acceso a estas funcionalidades es declarándolas explícitamente para que el sistema autorice a ello.

Cube´s War solo necesita de un permiso para su funcionamiento, que es el del control de hardware, en concreto, el control de vibración. Por lo tanto, se tiene que añadir ese permiso al *AndroidManifes.xml* mediante la etiqueta:

```
<uses permission android:name="android.permission.VIBRATE"/>
```

Código 4: Etiqueta Vibración

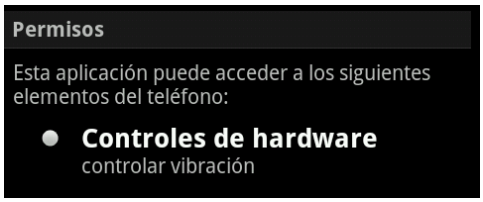


Figura 58. Parte de la ficha del juego instalado I

La implementación en código *Java* para que se produzca la vibración una vez añadido el permiso es la siguiente es sencilla.

Se realiza la comprobación de si el *boolean vibración*, contenido en la *Activity Proyecto*, se encuentra activado, y si es así, se produce la vibración del dispositivo tanto tiempo como reciba de la llamada del método. Los valores de tiempo que se establecen para determinadas acciones en el juego son de 50 y de 100 milisegundos. Las acciones de 50 ms. son las referenciadas a cuando el usuario es eliminado del juego siendo

indiferente la causa que lo haya provocado. Y la otra duración se produce cuando se ha logrado superar con éxito tanto un escenario como la totalidad del juego.

```
private void Vibrar(int tiempo)
{
   if (vibracion)
   {
      Vibrator vibrator
      =(Vibrator)getSystemService(Context.VIBRATOR_SERVICE);

      vibrator.vibrate(tiempo);
   }
}
```

Código 5: Vibración

4.3.3.3 Guardar configuraciones

Accediendo al menú de *Opciones* dentro del menú principal, se puede configurar al agrado del usuario la habilitación de la reproducción de efectos de sonidos o la vibración del dispositivo móvil ante distintas situaciones del juego, que más tarde se explicará.

Durante el transcurso del juego, estas condiciones se manejan mediante código, controlando las posibles situaciones que se pueden llegar a producir. Pero cuando la aplicación se ha cerrado, la configuración que había establecido el usuario se perderá, teniendo que volver a instaurarla cuando inicie de nuevo la aplicación.

También puede ocurrir que el usuario durante el transcurso de una partida, haya conseguido superar con éxito algún escenario de los que se presentan en el videojuego y decida posteriormente cerrar la aplicación. Más tarde cuando inicie de nuevo la aplicación y la partida, ese avance que había conseguido con anterioridad se perdía.

Ante estas situaciones, se incorporó a la aplicación el salvado de la configuración de sonido, vibración y el número del escenario en el que se encuentra cuando el usuario cierre la aplicación. Esto se realiza mediante la creación de un fichero llamado *Configuracion.txt*, ubicado en el directorio de sistema de ficheros donde residen los ficheros internos de la aplicación, es decir, en la memoria interna del dispositivo dedicada a la aplicación. Guardando la configuración en ese directorio y en modo *private*, se consigue que ni el usuario ni otra aplicación puedan acceder a ella siendo su uso restringido a la propia, y que sea eliminado automáticamente cuando se desinstale la aplicación del dispositivo móvil.

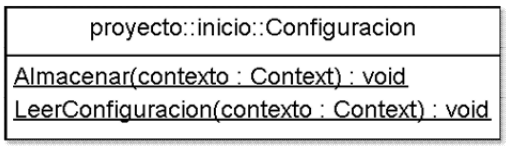


Figura 59. Clase *Configuración*

La técnica para conseguir este cometido se realiza en el método *Almacenar* de la clase *Configuración*, siguiendo estos pasos:

1. Obtener el *path* absoluto del directorio de sistema de ficheros donde residen los ficheros internos de la aplicación.

2. Acreditar si dentro de ese directorio anterior existe el fichero *Configuracion.txt* para su posible creación.

3. Abrir o crear el fichero y escribir en una única línea tres números. El primero es referente a la configuración del sonido, valiendo 0 si no está habilitado y 1 si lo está. El segundo trata de la configuración de la vibración cuyo valor se establece procediendo de igual modo que con el sonido. Por ejemplo, si el sonido esta habilitado y la vibración no, en el fichero se guardará el valor *10*. Y por último, el tercer valor corresponde al número de escenario en el que se encuentra el usuario antes de cerrar la aplicación, pudiendo contener valores desde 0 a 2, ambos inclusive.

   Cabe destacar que siempre que se abra el fichero, sobrescribirá todo su contenido debido a que al guardarse en modo *private*, pueden ocurrir dos cosas, o bien crea por primera vez el fichero o por el contrario reemplaza el archivo que tenga el mismo nombre. Esta característica es beneficiosa en el caso de esta aplicación, dado que la configuración del videojuego y el número de escenario en el que se encuentre, puede variar de una partida a otra.

4. Cerrar el fichero.

El archivo de configuración resultante, forma parte de los datos de la aplicación ocupando un tamaño de almacenamiento de *4,00 KB*, como se muestra en la siguiente imagen:

Figura 60. Parte de la ficha del juego instalado II

4.3.3.4 Cargar configuraciones

Cuando el usuario lanza la aplicación posteriormente al haberla ejecutado por primera vez, aparece el menú principal y de manera no visible a éste, se procede la carga de las configuraciones.

El lugar donde están ubicadas las configuraciones se ha descrito con anterioridad, por lo que se proceden a explicar los pasos que se realizan en el método *LeerConfiguracion* de la clase *Configuración*, para conseguir la carga de éstas:

1. Acceder al directorio específico y abrir el fichero *Configuracion.txt.*

2. Leer la única línea que contiene el fichero y guardarla en una variable.

3. Analizar los tres números que contiene la variable, teniendo en cuenta que el primero es referente al sonido, el segundo a la vibración y el tercero al número de escenario donde se abandonó la partida anterior. Con los números obtenidos se establecen los valores de los *booleans* que condicionan la reproducción de sonidos y la vibración, es decir, si los valores obtenidos son 1 respectivamente, valdrán *true* y al contrario si se son 0. Y por último se iguala el tercer valor a la variable *numeroEscenario* de la clase *Proyecto.*

4. Cerrar el fichero.

Puede darse el caso que el usuario emplee por primera vez el juego o que éste tenga un nivel de conocimiento mayor de la plataforma y elimine la configuración de la aplicación pulsando el botón de *Borrar Datos* como se muestra en la imagen anterior. Por lo tanto, la aplicación al acceder al directorio donde debe estar ubicado la configuración no encuentre fichero alguno. Ante esto, cuando se ejecuta la aplicación primero se establecen unos valores por defecto como son el de la habilitación de sonido y vibración y el número de escenario que se cargue es el 0, para posteriormente leer el fichero. Como no se encontrará fichero alguno, esos valores ya estarán inicializados y no hay que constituir de nuevo una configuración estándar.

4.3.3.5 Menú principal

Debido a que el diseño de una pantalla a través de código *Java* puede resultar muy complejo y un poco tedioso, *Android* proporciona la posibilidad de poder desarrollarlas de una manera más fácil por soportar sintaxis *XML* dedicado al diseño de pantallas. *Android* define un gran número de elementos especialmente hechos para la plataforma, cada uno de ellos representando una subclase específica de *Android View*[100]. Se puede diseñar una pantalla de la misma forma que se diseña en *HTML*[101], es decir, como una serie de etiquetas anidadas y guardadas en un archivo *XML* dentro del directorio *res/layout/* de la aplicación.

Cada etiqueta de los archivos *XML*, equivale a un único elemento *android.view.View*, pudiendo ser un simple elemento visual o un elemento de esquema que contiene una colección de objetos hijos. Estas etiquetas si se refieren a un elemento visual, se van añadiendo una detrás de otra pero se debe tener cuidado porque *Android* tiende a dibujar los elementos en el orden en el cual aparecen en el documento *XML*. Por lo tanto, si hay componentes que se sobreponen, el último en el archivo será probablemente el que aparezca sobre cualquier otro que esté previamente declarado en su mismo espacio. Existen etiquetas que sirven para ordenar de distinta manera estos elementos y conseguir que no se produzca el problema anteriormente citado. Son conocidas como *layouts*.

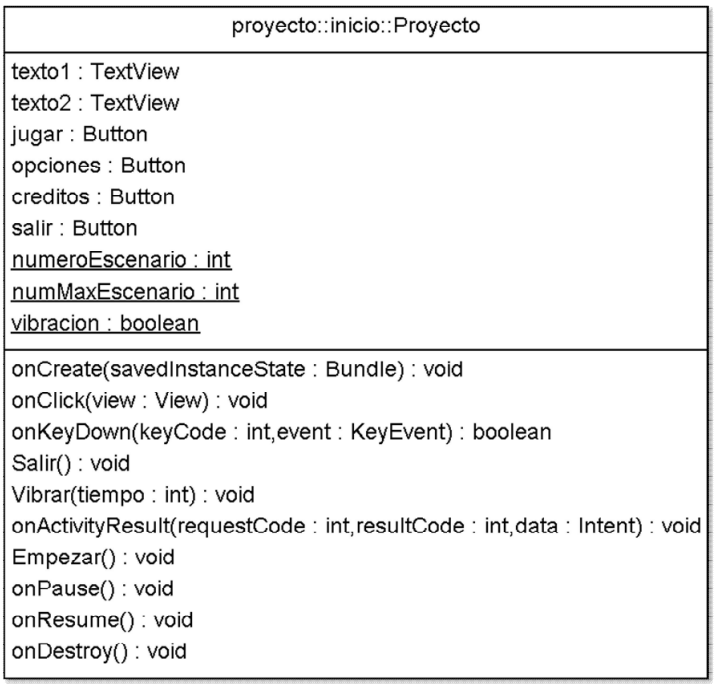


Figura 61. *Activity Proyecto*

En el caso de la aplicación *Cube´s War*, la *Activity Proyecto*, además de ser la vital, presenta la pantalla del menú principal que es a su vez la portada de bienvenida, utilizándose una serie de elementos de la *GUI* (interfaz gráfica) de forma que resulte atractiva al usuario. En concreto esta formado por dos *TextView* y cuatro *Buttons*, ordenados por un *LinearLayout* de forma que cada vez que se añade un elemento, estos se ordenen en forma vertical.

Figura 62. Menú principal

Cada archivo *XML* esta formado por etiquetas que corresponden a clases *GUI* de *Android*. Estas etiquetas tienen atributos que en su mayoría corresponden a métodos en código *Java*, y de las cuales se han utilizado un número determinado de ellas para alcanzar el resultado obtenido que se puede observar en la imagen superior. Algunas de las etiquetas utilizadas han servido para establecer el color del texto, la imagen de fondo, el espacio de sus márgenes, su identificador…

La configuración *XML* del menú principal se puede cargar en el código llamando al método *setContentView (R.layout.main)* dentro de la implementación que se haga del

método *onCreate()* de la *Activity*. Esto es debido a que cuando *Android* compila una aplicación, compila cada archivo en un *android.view.View*.

Una vez con la distribución cargada, también se pueden asociar los elementos de esa configuración *XML* a los propios del código *Java*, mediante el identificador asignado a ellos citado con anterioridad. Esta asignación se hace mediante el método *findViewById(R.id.view_nombre)*, consiguiendo de esta manera unir los elementos del fichero *XML* con los del código *Java*, y poder establecer una funcionalidad a cada botón para que accedan a otra actividad o que realicen una determinada acción.

El menú principal conforma una distribución de elementos mediante el uso de la sintaxis *XML* cargada en el código *Java*. Pero existen características determinadas de éste que merecen una explicación más detallada:

- Los botones del menú principal no se les asigna una imagen de fondo contenida en la carpeta *res/drawable/*, si no un fichero *XML*. Ese fichero *XML* define diferentes imágenes de fondo dependiendo del estado del botón, mediante el uso de *selectores*.

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">

        <item android:state_focused="true" android:state_pressed="false"
        android:drawable="@drawable/jugarf" />

        <item android:state_focused="true" android:state_pressed="true"
        android:drawable="@drawable/jugarp" />

        <item android:state_focused="false" android:state_pressed="true"
        android:drawable="@drawable/jugarp" />

        <item   android:drawable="@drawable/jugar" />
</selector>
```

Código 6: *XML* de botones

Un *selector* se define mediante un fichero *XML* localizado en la carpeta *res/drawable*, y en él se pueden establecer los diferentes valores de una propiedad determinada de un control dependiendo de su estado. Por lo tanto, los botones tendrán distintas imágenes dependiendo de si tienen el foco, están pulsadas o se encuentran en estado normal.

Como ejemplo de esta explicación, se muestran las distintas imágenes que puede tener el botón Jugar del menú principal, al igual que los distintos botones, diferenciándose solamente en el texto de la imagen.

Figura 63. Imagen del botón Jugar

Figura 64. Imagen del botón Jugar con foco

Figura 65. Imagen del botón Jugar pulsado

- También referidos a los elementos del menú principal se destaca otra característica. Ésta es que a cada uno de ellos se le ha asignado una animación. Existen tres distintos tipos de animaciones en *Android*: *alpha*, *translate* y *rotate*. En la aplicación solo se han utilizado las dos primeras. *Alpha* permite hacer desaparecer o aparecer gradualmente el elemento asociado en base a la configuración y *translate* mueve los elementos de una posición inicial a la que tengan asociada en la configuración *XML*.

  Las animaciones se pueden definir en ficheros *XML* y se almacenan en la carpeta *res/anim.* Estas se asocian a los elementos que se quieran y se ponen en marcha del siguiente modo:

```
Animation a = AnimationUtils.loadAnimation(this, R.anim.translate);
a.reset();
jugar = (Button)findViewById(R.id.jugar);
jugar.clearAnimation();
Jugar.startAnimation(a);
```

Código 7: Asociación de animación

Este método se ha realizado de forma equivalente a cada uno de los elementos que tienen asignadas animaciones, de tal manera que se inician solo cuando la aplicación es ejecutada, obteniéndose un resultado sincronizado.

Por lo tanto, para el *TextView* del menú principal que contiene el nombre de la aplicación, se le asocia una animación de tipo *alpha* con el que se consigue una aparición gradual en un tiempo de 4,5seg, pudiéndose apreciar en el siguiente fragmento de código.

```
<?xml version="1.0" encoding="utf-8"?>
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
        android:fromAlpha="0.0" android:toAlpha="1.0"
         android:duration="4500">
</alpha>
```

Código 8: *alpha.xml*

Los botones *Jugar* y *Créditos* tienen asignados la misma animación de tipo *translate*, moviéndose de una posición -200 en el eje X, es decir, fuera de la zona de visualización de la pantalla, para detenerse en el centro de ésta, que es la posición determinada en la distribución *XML*.

```
<?xml version="1.0" encoding="utf-8"?>
<translate xmlns:android="http://schemas.android.com/apk/res/android"
        android:fromXDelta="-200%" android:toXDelta="0%"
        android:fromYDelta="0%" android:toYDelta="0%"
        android:duration="3500">
</translate>
```

Código 9: *translate.xml*

Y finalmente los botones *Opciones* y *Salir*, al igual que los anteriores tienen la misma animación asignada, pero en lugar de ser el desplazamiento de izquierda

a derecha, es al contrario, de una posición +200 en el eje X a su posición central determinada en la configuración *XML*.

```
<?xml version="1.0" encoding="utf-8"?>
<translate xmlns:android="http://schemas.android.com/apk/res/android"
        android:fromXDelta="200%" android:toXDelta="0%"
        android:fromYDelta="0%" android:toYDelta="0%"
        android:duration="3500">
</translate>
```

Código 10: *translate2.xml*

Se debe mencionar que cada uno de los *TextViews* que se observen en la aplicación, tanto en pantallas emergentes como en menús, obtienen su contenido de un recurso. *Android* permite definir *Strings* en uno o más archivos *XML* de recursos. Estos archivos están bajo el directorio *res/values*, siendo su nombre por convención *strings.xml*. Se acceden al contenido de cada uno de los *Strings* que se han definido dentro de estos ficheros, mediante la llamada *setText(R.string.nombre)*. Uno de estos *TextView* que se encuentra en la parte inferior derecha del menú principal varía su contenido dependiendo del estado de habilitación del sonido, obteniendo el texto que debe mostrar al igual que la explicación anterior.

Por lo tanto, como resultado de todo esto es que cuando se lanza la aplicación, el usuario observa como se desplazan los componentes del menú debido a la animación asociada, hasta que esta finaliza, pudiendo posteriormente realizar cualquier acción que se desee, siendo éstas las siguientes:

- **BOTÓN SALIR**

El botón *Salir* realiza la acción de cerrar la aplicación. Para ello se presenta como paso intermedio un *Dialog* personalizado, el cual sigue la estética de colores y fuentes del videojuego. La distribución de los elementos de la pantalla es cargada desde el archivo *XML dialogo_salir.xml*. Contiene el icono de la aplicación, un *TextView* con información y dos *Buttons* con las opciones posibles además de un escuchador de eventos asociado. Asimismo el *Dialog* tiene asignado uno de botones físicos, para que de este modo si se pulsa el botón físico ⮌ del dispositivo móvil, nos conduzca de nuevo al menú principal pasando del estado *onPause* cuando el *Dialog* esté visible, a *onResume* cuando se cierre. Todo esto es debido a que como se ha explicado anteriormente, se implementa en el código estos estados para su manejo.

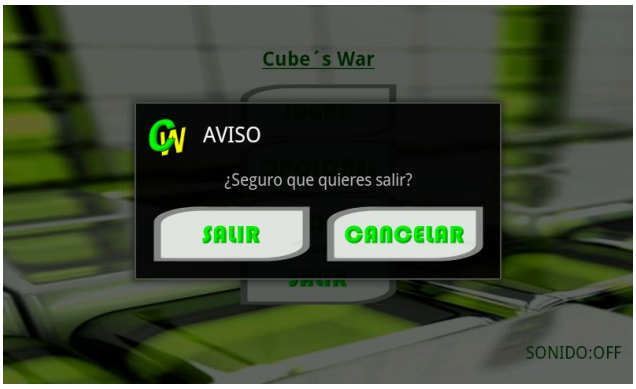


Figura 66. Diálogo salir

Los botones tienen distintas imágenes de fondo dependiendo de cual es su estado, al igual que se establece en el menú principal. El botón *Salir* del diálogo, llama al método necesario para guardar la configuración del sonido, de la vibración y del número de escenario en el que se abandonó la partida, además de cerrar la aplicación y el diálogo. Por otro lado, el botón *Cancelar* retorna al menú principal que es de donde se procedía, cerrando la ventana.

- **BOTÓN CRÉDITOS**

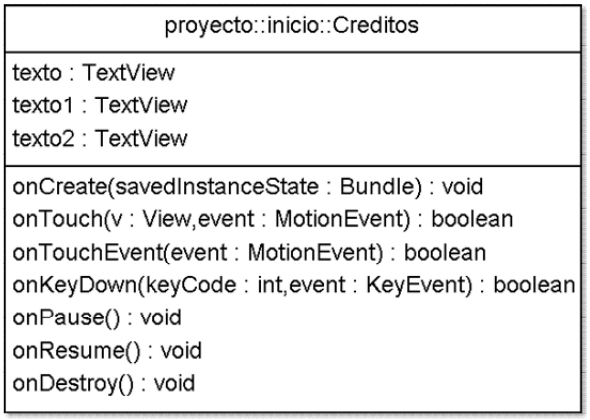


Figura 67. *Activity Créditos*

El botón *Créditos* del menú principal nos conduce hacia otra *Activity* en el que se muestra en una pantalla los autores del videojuego. La *Activity* principal no finaliza ni se cierra, si no que queda en segundo plano, es decir, pausada, controlándose su nuevo estado debido a la implementación del método *onPause* por parte de ésta. Con esto se consigue que cuando se quiera volver al menú inicial, solo baste con finalizar la *Activity Créditos*, y pasar al primer plano la *Activity Proyecto*. Si se decidiera cerrar ésta ultima y volver a crearla cuando se cerraran los *Créditos,* se observaría de nuevo la animación de colocación de los elementos del menú principal que se produce al iniciar la aplicación, por lo que retrasa el tiempo de ejecución del usuario.

Figura 68. Créditos de la aplicación

Como se puede observar en el resultado obtenido, la *Activity* no ocupa toda la pantalla y su apariencia es similar a la de un *Dialog*, como el comentado anteriormente. Esta peculiaridad se consigue modificando el fichero *AndroidManifest.xml*, añadiendo en la etiqueta de la *Activity Créditos* el siguiente atributo de estilo:

```
android:theme="@android:style/Theme.Dialog"
```

Código 11: Atributo Estilo

La distribución está determinada por el recurso *XML creditos.xml* y los contenidos de los *TextViews* cargados a partir del archivo s*trings.xml* ubicado en *res/values.*

Esta *Activity* y los *TextViews* que la conforman, tienen asociado un escuchador de eventos de Contacto, para que cuando se realice alguna acción sobre algunos de ellos, ésta finalice y por lo tanto la pantalla con los autores. Además del escuchador de eventos anterior, la *Activity* tiene asignado uno de teclado por si se pulsa el botón físico ⮌ del dispositivo móvil, realizando la misma acción que la anteriormente mencionada. Cuando se produzcan algunas de las acciones citadas, la *Activity* principal que se encontraba *onPause*, pasará al estado de *onResume*.

- **BOTÓN OPCIONES**

En esta sección permite al usuario poder habilitar a o no la reproducción de sonidos y la posibilidad de que el dispositivo móvil vibre ante distintas acciones que ya se han mencionado. Cuando se seleccione dentro del menú principal *Opciones*, aparece la siguiente ventana:

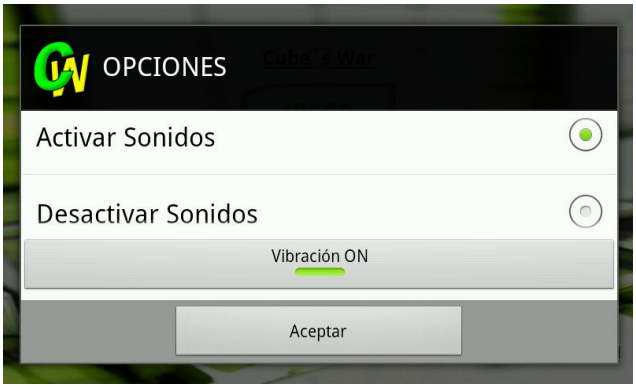


Figura 69. Opciones de la aplicación

Al contrario que las demás pantallas que se han analizado, esta no carga su distribución mediante un recurso *XML*, si no que gracias a *AlertDialog.Builder* que pone a disposición *Android*, se puede ir añadiendo distintos elementos a la ventana hasta que no se produzca la llamada de *show* que provoca su visualización. Las características más comunes en un *AlertDialog.Builder* es la aparición de un titulo, un mensaje de texto, uno, dos o tres botones y una lista de elementos seleccionables.

Siguiendo el orden que se puede observar en la imagen superior del menú de *Opciones*, está compuesto por los siguientes elementos:

- una lista de elementos seleccionables denominada *SingleChoiceItems*. Este se caracteriza por poder escoger solo un elemento dentro de esa lista que se le proporciona, utilizando este recurso para la des/activación del sonido. Antes de la visualización de la ventana, se analiza el *boolean* que controla el estado de habilitación del sonido, para que de este modo se muestre marcado el *RadioButton* correspondiente a su estado.

- un *ToggleButton* que permite establecer si quiere que la aplicación vibre ante determinadas acciones. Este elemento muestra su estado de selección o de no selección como un botón con una luz indicadora acompañado de un texto que varia cuando su estado está o no activado. Para este caso determinado, el texto cuando la luz indicadora está iluminada es de *Vibración ON*, mientras que si no lo está es *Vibración OFF*. Al igual que el caso anterior, se analiza si la vibración esta habilitada para que cuando se produzca su visualización, se muestre en el estado adecuado.

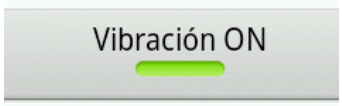


Figura 70. Estado ON del ToggleButton

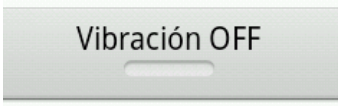


Figura 71. Estado OFF del ToggleButton

- un *PositiveButton* con el texto *Aceptar* que al ser pulsado, determina que elemento está seleccionado en *SingleChoiceElements* para cambiar la habilitación o no del sonido. También comprueba si el *ToggleButton* se encuentra activado y en función de esto, configura la activación de la vibración. Y por ultimo, cambia el texto del menú principal que indica si el sonido está activado o desactivado y cierra la ventana.

Esta ventana tiene asociado un escuchador de eventos de teclado ofreciendo la posibilidad al usuario de que si pulsa el botón físico ⮌ del dispositivo móvil, pueda retroceder al menú principal sin que se establezca la configuración que había realizado en el menú de Opciones.

- **BOTÓN JUGAR**

Con la selección de esta opción, el usuario iniciará la partida al juego, por lo que toda su lógica se pondrá en marcha.

Para ello, *Proyecto* inicia una nueva *Activity* llamada *Videojuego*, asociándole un código representativo a ésta, quedando a la espera de recibir resultados que *Videojuego* debe devolver cuando finalice, explicándose más adelante, lográndose del siguiente modo:

```
Intent nuevoIntent = new Intent(this,Videojuego.class)

this.startActivityForResult(nuevoIntent, 1111)
```

Código 12: Inicio *Activity Videjuego*

La *Activity Proyecto* pasa a segundo plano en modo pausa gracias a la implementación por parte de ésta del método *onPause*, y porque *Android* trata las *Activities* como si fuera una pila colocándolas una encima de otra, como ya se ha dicho con anterioridad. Actuando de este modo, cuando finalice la *Activity Videojuego*, se visualizará *Proyecto* dado que se encontraba esperando a volver al primer plano, lográndose mediante el método *onResume*. Si se decidiera cerrar *Proyecto*, cuando finalizara *Videojuego* se visualizaría de nuevo la animación inicial, por lo que produce un retraso de actuación al usuario.

Por lo tanto, el botón *Jugar* inicia una nueva *Activity*, con esto se produce una carga de elementos, entorno de desarrollo, lógicas… pero antes de describen cada una de las clases que conforman el proyecto para un mejor entendimiento.

4.3.3.6 Clases

ELEMENTO

Esta clase no pertenece a este proyecto si no que forma parte de la biblioteca desarrollada por otro PFC de una compañera. Se cita debido a que es una clase importante en el conjunto del videojuego y se caracteriza por ser de tipo abstracta, usándose únicamente para definir un conjunto de subclases, como las que forman parte del presente proyecto, siendo *LogicaProtagonista*, *LogicaEnemigo*, *LogicaMeteorito* y *Cámara*. Contiene una serie de métodos que no tienen implementación, es decir, son abstractos y que por lo tanto, las clases mencionadas deben implementar.

Todo lo referente a los movimientos de los elementos, a la comprobación de las colisiones del personaje, enemigos y meteoritos, cámara del juego, está contenido en sus respectivas clases, y por parte de la librería implementada por otro PFC, se encarga de asociar un objeto tridimensional a estos, para su dibujado y el cambio de la imágenes *Sprites* que tienen asociada.

- **LOGICAPROTAGONISTA**

Clase que hereda de *DibujaProtagonista*, y ésta de *Elemento*, ambas clases pertenecientes a la biblioteca utilizada en este proyecto. Implementa los métodos que en éste se han declarado como abstractos. Esta clase maneja todo lo relacionado con el personaje del videojuego como los movimientos tanto en el eje X, Z e Y, su posición y la detección de las colisiones que se pueden producir.

Como se puede observar más adelante, contiene muchas variables con distinto significado, procediendo seguidamente a su explicación:

- ***private float posX, posY, posZ*:** posiciones en los ejes del personaje que controla el usuario. Solo se realiza una única operación sobre ellos, porque las variables copia de éstos son los que sufren las operaciones que se efectúan, para posteriormente copiar sus valores. Esto se realiza con el fin de proteger a estos datos de posibles errores que se puedan producir.

- ***float dX, dY, dZ*:** las copias de las variables sobre las que se realizan las operaciones.

- ***private boolean izquierda, derecha, arriba, abajo*:** habilitados cuando se pulsa sobre los píxeles de la pantalla correspondientes a la posición de los botones del pad, respectivamente. Se usan para establecer que el usuario quiere mover al personaje en el sentido que indican sus nombres.

- ***private boolean poderIzq, poderDer, poderAr, poderAb*:** solo habilitados cuando sus homólogos anteriores lo están, y después de haber determinado que

en el sentido que quiere ir el usuario no hay obstáculo alguno sobre el escenario en el que se encuentra, es decir, no hay colisiones.

Figura 72. Clase *LogicaProtagonista*

- ***private static boolean poderSaltar, poderSubir, poderBajar, saltoTerminado*:** variables que se usan cuando el usuario quiere que el personaje salte. El primero

determina si se puede saltar. El segundo si una vez que se está saltando y se encuentra en la parte ascendente de éste, puede seguir subiendo si el escenario lo permite. Lo mismo sucede con el tercer valor, pero cuando el salto se encuentra en la fase descendente. Y la última indica que el salto ha terminado por encontrarse el personaje con plataforma debajo de su posición, y si no es así, entra en acción el método *Caer* que provoca el descenso del personaje gradualmente.

- ***private final float velocidadZ, velocidadX, V0, gravedad*:** variables que se utilizan para determinar cuánto se desplaza el personaje en el eje Z cuando se ha pulsado los pixeles correspondiente de la pantalla sobre los que se encuentran los botones que realizan esa acción, o cuánto se desplaza en el eje X, o con la velocidad inicial con la que el personaje comienza el salto y la gravedad que soporta durante éste.

- ***private int dedo*:** variable que guarda el identificador que devuelve *Android* para diferenciar el elemento que ha realizado la pulsación del botón de *Saltar* sobre los demás.

- ***public static boolean llaves*:** establece si el usuario, mediante el movimiento del personaje, ha estado en la misma posición de la llave dispuesta en el escenario. Si es así, se permite el avance al siguiente cuando se llegue al final del escenario. Su valor inicial es *false* y se reinicia cuando se produce el hecho anterior. Si el personaje es eliminado y había conseguido la llave, se mantendrá esa condición para no volver a tener que obtenerla.

- ***private static int clicks*:** variable que se incrementa conforme las veces que se ejecuta el bucle principal del juego. Se utiliza como unidad de tiempo para la acción *Saltar*. Ésta, al basarse en formulas físicas del tiro vertical, necesita una variable de tiempo que se vaya incrementando para que de este modo avance el estado del salto. Una vez finalizado, se igualará a 0 y hasta que el usuario no pulse de nuevo el botón *Saltar*, no volverá a incrementarse.

Tres son los métodos que se utilizan para detectar el tipo de movimiento que se quiere hacer, dependiendo de los píxeles pulsados correspondientes al área de colocación de los botones. Sus nombres son iguales a los de los eventos que se producen sobre la pantalla del dispositivo móvil, convocados cuando son capturados.

- ***Down(boolean sentido)*:** recibe un boolean que determina el sentido del movimiento en el eje Z. Dependiendo de su valor habilita el *boolean abajo* o *arriba*.

- ***Move (int mov, int PointerID, Context contexto)*:** mediante la variable *mov* que se recibe se comprueba qué botón se ha pulsado. Si su valor es 1, se ha pulsado el botón de la izquierda del pad habilitando el *boolean* izquierda. Si es 2, ha sido el botón de la derecha habilitando *derecha*, y finalmente si es 3, significa que se ha pulsado sobre el botón de *Saltar*. En este caso se utiliza la variable *PointerID* para guardarla en la variable global *dedo*, determinando de este modo el identificador del dedo que ha realizado esa pulsación, dado que hasta que no

se vuelva a levantar ese dedo sobre la pantalla, no se vuelve a habilitar el salto del personaje.

- ***Up(int idDedo)*:** se recibe el identificador del dedo que se ha levantado de la pantalla. Si es igual esa variable a la global *dedo, poderSaltar* se habilita. Esto se realiza con el fin de que no produzca el salto del protagonista más de una vez por pulsación del botón *Saltar*, solo debe realizarse una vez por pulsación.

Posteriormente en el método *Colisiones*, mediante la comprobación de las variables *izquierda, derecha, arriba, abajo, saltoTerminado* y consecutivamente si el tipo de movimiento que se quiere realizar con el personaje, determinado por el nombre de la variable, se verifica trasladando la posición de éste sobre la matriz del escenario correspondiente, se habilitan las variables *poderIzq, poderDer, poderAr* y *poderAb.*

Con estas últimas variables en el método *Mover*, se constata su habilitación y dependiendo del caso, la posición del personaje *dX, dY, dZ* se incrementará o disminuirá en el eje correspondiente a los pixeles pulsados bajo los que se encuentra el botón que realice la determinada acción.

En el método *Actualizar* se copia los valores *dX, dY* y *dZ* que se han modificado a los de *posX, posY* y *posZ*, los cuales tiene un método asociado por cada eje para devolver su valor, llamados *damePosX*, *damePosY* y *damePosZ*.

- **SALTAR**

Los movimientos en los ejes X y Z del personaje como se ha visto anteriormente, solo se suma cierto valor a su posición en el eje cuando se han superado todas las comprobaciones pertinentes. Pero simular un salto basado en formulas físicas en un juego supone cierta complejidad.

El salto del personaje se basa en la teoría del "*tiro vertical*"[102], es un movimiento sujeto a la aceleración gravitacional, y es la aceleración la que se opone el movimiento inicial del objeto. Comprende subida y bajada de los cuerpos u objetos y se caracteriza por:

- La velocidad inicial nunca es cero.

- Mientras el objeto está ascendiendo el signo de la velocidad es positivo y la velocidad es cero en su altura máxima, cuando comienza el descenso el signo de la velocidad es negativo.

- La velocidad de subida es igual a la de bajada pero el signo de la velocidad al descender es negativo.

Se muestra a continuación las dos fórmulas que se utilizan para su cálculo, el significado de las siglas de esas fórmulas, sus variables correspondientes en la clase *LogicaProtagonista* que ya han sido explicadas con anterioridad además de su funcionalidad, y si sus valores son fijos o pueden variar:

FORMULAS	SIGNIFICADO	VARIABLE	TIPO
$V_f = V_0 - (g * t)$	$V_f \equiv$ Velocidad Final	*Vf*	*static*
	$V_0 \equiv$ Velocidad Inicial	*V0* = 0.8	*final*
	g ≡ Gravedad	*gravedad* = 5.0	*final*
$h = (V_f^2 - V_0^2) / (- 2 * g)$	t ≡ Tiempo	*clicks*	*static*
	h ≡ altura	*salto*	*static*

Tabla 48: Especificaciones del salto

Los pasos para efectuar el salto real del personaje son los siguientes:

1. Se realiza el cálculo de *Vf* en la primera fórmula utilizando las variables necesarias de la clase.

2. Se determina *salto* en la segunda, haciendo uso del valor de *Vf* determinado con anterioridad y demás variables necesarias.

3. Se analiza si *Vf* tiene un valor positivo o negativo. Si es positivo se encuentra en su momento de subida y si es negativo, en el de bajada, como se ha comentado anteriormente, realizando distintos cálculos en función de su signo:

   - **Si es positivo:** se comprueba que la *posY* del personaje no supera las dimensiones en el eje Y del escenario porque si se cumple no se permite ascender al protagonista. Posteriormente, se prueba si la *posY* del personaje más la variable *salto* calculada en el paso 2, se encontraría con algún obstáculo del escenario, porque si es así se impediría seguir ascendiendo en el salto al personaje. Si pasa esas dos comprobaciones, se resta *salto* a la posición en el eje Y del personaje.

   - **Si es negativo:** se comprueba si la *posY* del personaje menos la variable *salto* existe plataforma del escenario, porque si es así, se daría por finalizado el salto del personaje y si restaría *salto* a la posición en el eje Y del protagonista.

Todos estos pasos se realizan de forma recurrente en el bucle principal del juego, y no se da por finalizado el salto hasta que el personaje encuentre plataforma bajo él, por lo que cada vez que se realizan todos los pasos anteriores se avanza un poco en el estado del salto.

- **LOGICAENEMIGO**

Clase que hereda de *DibujaEnemigo*, y ésta de *Elemento*, ambas clases pertenecientes a la biblioteca utilizada en este proyecto. Implementa los métodos que en éste se han declarado como abstractos.

Esta clase maneja todo lo relacionado con los enemigos del videojuego como los movimientos tanto en el eje X, Z pero no en el Y que siempre será la misma, su posición y la detección de la colisiones que se pueden producir.

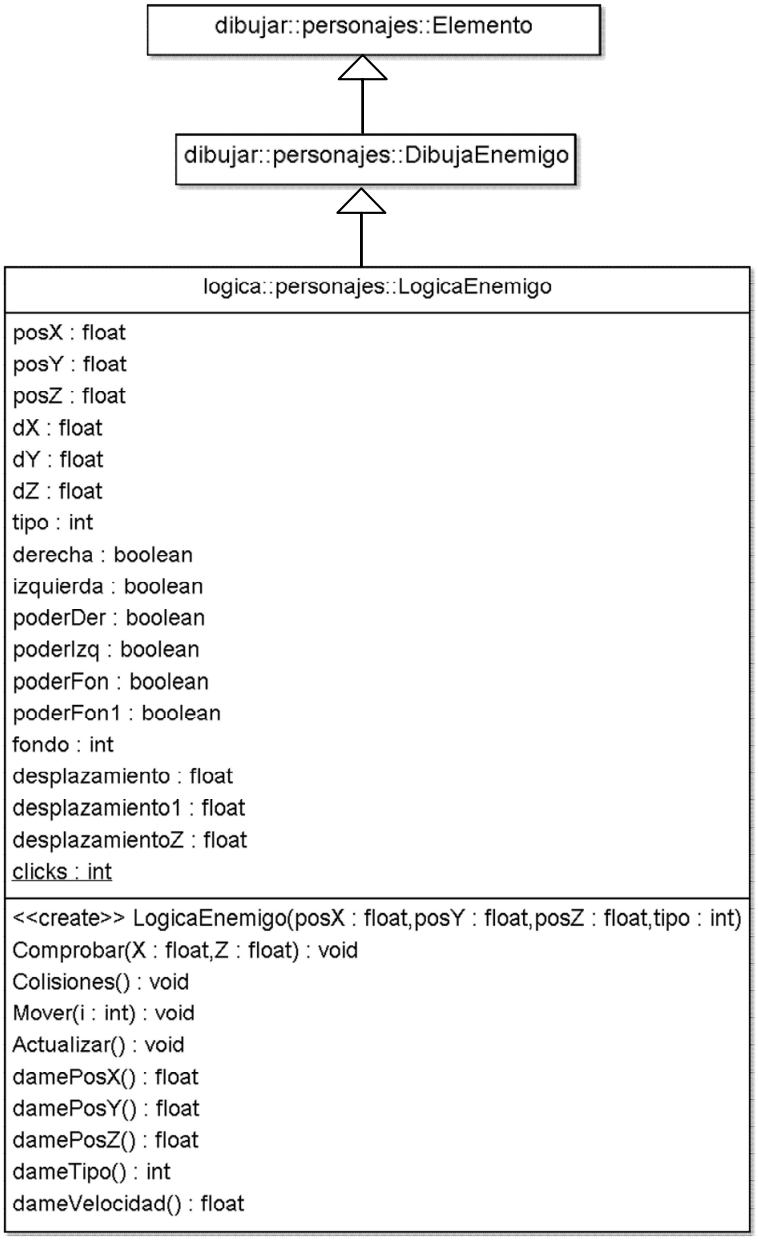


Figura 73. Clase *LogicaEnemigo*

Tiene muchas variables con distinto significado, procediendo seguidamente a su explicación:

- ***float posX, posY, posZ*:** posiciones en los ejes de los enemigos. Solo se realiza una única operación sobre ellos, porque las variables copia de éstos son los que sufren las operaciones que se efectúan, para posteriormente copiar sus valores. Esto se realiza con el fin de proteger a estos datos de posibles errores que se puedan producir.

- ***private float dX, dY, dZ*:** copias de las variables sobre las que se realizan las operaciones.

- ***private int tipo*:** determina el tipo de *Enemigo* que es. Si es de tipo 0 solo se mueve en el eje de las X de manera lenta. Si es de tipo 1, es idéntico al del tipo 0

pero se desplaza algo más rápido. Y finalmente si es de tipo 2, se mueve con la misma velocidad que los de tipo 0, pero con la peculiaridad de que también se puede desplazar en el eje Z. Cada uno de estos tipos tiene su propia imagen asociada y por lo tanto *Sprite*.

	TIPO 0	TIPO 1	TIPO 2
IMAGEN			

Tabla 49: Imágenes asociadas a los distintos tipos de enemigos

- ***private boolean derecha, izquierda*:** habilitados cuando mediante cálculos sobre la posición del protagonista y la suya, se determina que el *Enemigo* debe desplazarse hacia el sentido que indica el nombre de la variable.

- ***private int fondo*:** su valor puede ser 0, 1 o 2 y solo está permitido para los *Enemigos* de tipo 2. Representa si la posición Z del *Enemigo* es igual "0", inferior"1" o superior "2" a la del *Protagonista.*

- ***private boolean poderDer, poderIzq, poderFon, poderFon1*:** solo habilitados cuando sus homólogos anteriores lo están, y después de haber determinado que en el sentido que debe ir el *Enemigo* no hay obstáculo alguno sobre el escenario en el que se encuentra o no hay plataforma donde apoyarse, es decir, no hay colisiones. *poderFon* y *poderFon1* solo están disponibles para los *Enemigos* tipo 2, porque indica que éste se puede desplazar en el eje Z.

- ***private final float desplazamiento, desplazamiento1, desplazamientoZ*:** valores que representan cuánto se puede desplazar un *Enemigo* en el eje X y solo en el eje Z para aquellos que sean de tipo 2.

- ***private static int clicks*:** variable que se incrementa conforme las veces que se ejecuta el bucle principal del juego, hasta un valor determinado. Hasta que no suceda este hecho, indica que no puede desplazarse el enemigo tipo 2 en el eje Z. Es a modo de retardo dado que si no el desplazamiento en ese eje sería tan rápido, que el personaje no podría avanzar en el escenario al encontrarse con dicho enemigo. Al obtener el valor máximo, se produce el movimiento en el eje Z, se vuelve a igualar a 0, y hasta que el enemigo no detecte que su posición Z es distinta a la del protagonista, no volverá a incrementarse.

Este método *Comprobar(float X, float Z)* establece en qué sentido debe desplazarse el *Enemigo* en función de las variables que recibe, que son la posición en el eje X y en el eje Z del *Protagonista,* habilitando, dependiendo del caso y tipo de *Enemigo*, *derecha, izquierda* o *tipo*.

Posteriormente en el método *Colisiones* mediante la comprobación de las variables *derecha, izquierda, tipo* y consecutivamente si el tipo de movimiento que quiere realizar el *Enemigo,* determinado por el nombre de la variable, se verifica trasladando la

posición de éste sobre la matriz del escenario correspondiente, se habilitan las variables *poderDer*, *poderIzq*, *poderFon* y *poderFon1*.

Con estas últimas variables en el método *Mover*, se constata su habilitación y dependiendo del caso, la posición de *Enemigo dX, dZ* se incrementará o disminuirá en el eje correspondiente. Cabe mencionar que los movimientos en el eje Z de los *Enemigos* tipo 2, tienen asociado una variables que hasta que no se entra 55 veces en ese método, no se realiza el movimiento.

Por último en el método *Actualizar* se copia los valores *dX, dY* y *dZ* que se han modificado a los de *posX, posY* y *posZ.*

- **LOGICAMETEORITO**

Clase que hereda de *DibujaObjeto*, y a su vez de *Elemento*, ambas clases pertenecientes a la biblioteca utilizada en este proyecto implementando los métodos que en ellos se han declarado como abstractos. Esta clase maneja todo lo relacionado con los meteoritos del videojuego como los movimientos en el eje Y dado que sus posiciones X y Z siempre serán las mismas, su posición y la detección de las colisiones que se pueden producir.

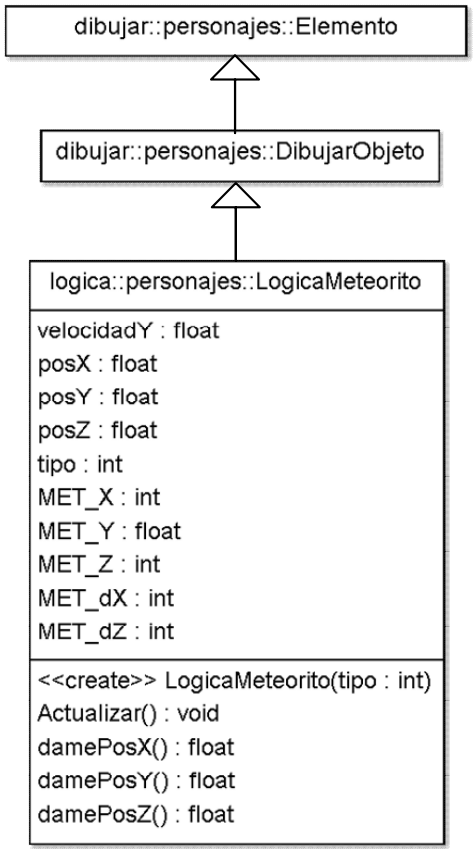


Figura 74. Clase *LogicaMeteorito*

Tiene muchas variables con distinto significado, procediendo seguidamente a su explicación:

- ***private float posX, posY, posZ*:** posiciones en los ejes de los meteoritos.

- ***private final int MET_X, MET_Z*:** posiciones iniciales que se igualan a *posX* y *posZ* cuando se crea un objeto de esta clase. Son finales debido a que son fijas sus posiciones en esos ejes.

- ***private int tipo*:** determina si el meteorito creado es el que se encuentra en una posición más adelantada en el escenario si su valor es 0, y 1 si su posición es más retrasada.

- ***private float MET_Y*:** posición inicial en el eje Y. Su valor varía en cada momento restando a éste un factor determinado.

- ***private final int MET_dX, MET_dZ*:** debido a que son dos el número de meteoritos existentes por escenario, estas variables son el factor que se le restan a *MET_X* y *MET_Z* para que sus posiciones no sean las mismas en esos ejes.

- ***private final float velocidadY*:** factor que determina cuánto desciende en cada momento el objeto *LogicaMeteorito*, restándoselo a *posY*.

Existe un único método llamado *actualizaMeteorito*. Este modifica la *posY* de los meteoritos, restando *velocidadY* a esa coordenada. Antes de esa operación, en la posición de *mundo[posX][posY][posZ]* se iguala a 0, para producir el efecto de que a su paso destruye el escenario. Solo se realizará mientras que *posY* sea superior a 0 dado que puede conducir a errores en la matriz. Restar el factor a *posY* se seguirá realizándose hasta que su valor sea inferior a -9, es decir, que haya salido de la pantalla del dispositivo móvil, y cuando se cumpla esa condición se iguala *posY* a *MET_Y*, dado que es su posición inicial en ese eje.

- **CAMARA**

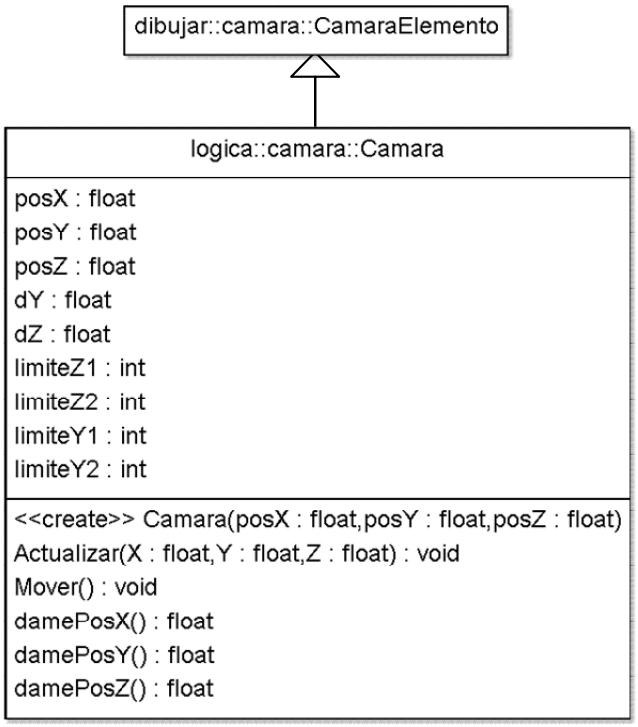


Figura 75. Clase *Cámara*

Clase que hereda de *CamaraElemento*, perteneciente a la biblioteca utilizada en este proyecto, y se utiliza para mover la cámara que enfoca el entorno tridimensional del videojuego. Para esta función, obtiene y trata los datos proporcionados por el sensor de Orientación del dispositivo móvil, que posteriormente será explicado.

Estos datos, que son dos, corresponden a cuánto se encuentra volteado el dispositivo móvil con respecto a sus ejes. De esto modo, se emplean sobre el videojuego para que cuando el usuario gire el dispositivo móvil en un sentido, el entorno donde se desarrolla el juego, también lo haga. El fin de esto es que al ser un videojuego de plataformas con la característica de poder desplazar al personaje en el eje Z, se pueda cambiar la perspectiva y el grado de visualización del usuario, consiguiendo que éste observe si puede avanzar en ese sentido.

Para todo este cometido se emplean en la clase *Cámara*, un conjunto de variables que son:

- ***public float posX, posY, posZ*:** propiedades de la *Cámara*. Solo se realiza una única operación sobre ellos, porque las variables copia de éstos son los que sufren las operaciones que se efectúan, para posteriormente copiar sus valores. Esto se realiza con el fin de proteger a estos datos de posibles errores que se puedan producir. El primer valor es la posición en el eje X de la *Cámara* que corresponde a la del *Protagonista*, debido a que ésta siempre debe enfocarle. Los demás valores son los que devuelve el sensor de Orientación, una vez ya tratados. El movimiento de la cámara se explica en detalle más adelante.

- ***private float dY,dZ*:** copias de las variables sobre las que se realizan las operaciones. Son las correspondientes a los valores del sensor, por ser estos los que deben ser modificados.

- ***private final int limiteY1, limiteY2, limiteZ1, limiteZ2*:** límites inferior y superior de giro del entorno de desarrollo del videojuego sobre cada eje respectivamente.

Debido a que casi de manera constante cambian los valores del sensor y la posición X del *Protagonista*, el método *Actualizar(float X, float Y, float Z)* recibe esos cambios y los guarda en las variables globales *posX*, *dY* y *dZ* respectivamente.

Se ha comentado con anterioridad que existen unas variables que limitan el giro del entorno del videojuego con respecto al del dispositivo móvil. En el método *Mover()* es donde se imponen, dado que si *dY* y *dZ* son mayores al limite superior del eje o menores al inferior, se igualaran a estos, para posteriormente copiar su valor a las variables globales *posY* y *posZ*.

Estos valores y límites son los que se consideran adecuados para este videojuego en concreto, girando el entorno *posY* y *posZ*, o desplazándolo, *posX*.

- **ESCENARIOS**

Los escenarios en este proyecto corresponden a una matriz tridimensional de números enteros llamada *mundo[][][]* de la clase *Escenario* (clase no perteneciente a este

proyecto). Cada una de las dimensiones atañe a los ejes X, Y, Z del escenario, cuyos valores máximos vienen determinados por las variables *MAX_X*, *MAX_Y* y *MAX_Z* de la clase *Lógica* perteneciente a este proyecto y que posteriormente será explicada.

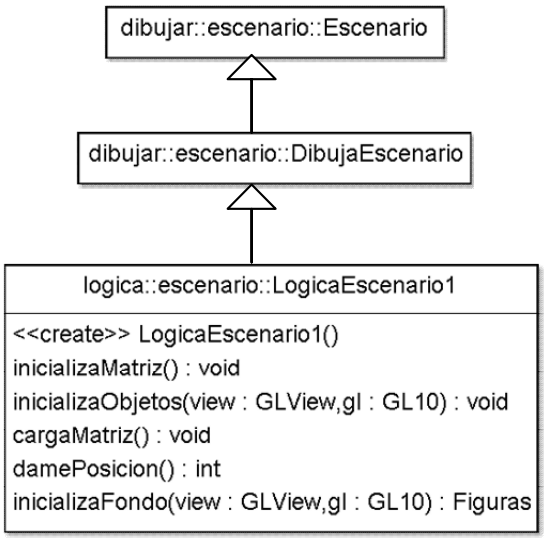


Figura 76. Clase *LogicaEscenario1*

Cada uno de los valores almacenados es un número entero, que corresponde a que en ese escenario en la posición X, Y, Z existe un determinado elemento no móvil, es decir, estático, dependiendo de su valor, cuyo significado es el siguiente:

- **"0":** no existe plataforma en esa posición del escenario.
- **"1":** plataforma sobre la que el personaje podrá desplazarse o colisionar.
- **"2":** elemento puntiagudo sobre el que el personaje no debe caer.
- **"3":** donde se encuentra la meta del escenario. Todos los escenarios tienen el final en las mismas coordenadas, determinado por las variables enteras *NAVE_X*, *NAVE_Y* y *NAVE_Z* de la clase *Lógica*.
- **"4":** establece que en esa posición se localiza la llave por las variables *LLAVE_X*, *LLAVE_Y*, *LLAVE_Z*, también de la clase *Lógica* la cual se debe recoger para poder pasar al siguiente escenario.

Tres son las clases en las que se crean y manejan los escenarios: *LogicaEscenario1*, *LogicaEscenario2* y *LogicaEscenario3*.

LogicaEscenario1, es una clase que hereda de otra intermedia que a su vez lo hace de *Escenario*, por lo que sus variables estarán disponibles por parte de esta clase y los métodos declarados como abstractos los debe implementar. *LogicaEscenario2* y *LogicaEscenari3* conforman el mismo caso que *LogicaEscenario1*. Ésta corresponde al primer escenario del videojuego, *LogicaEscenario2* al segundo y *LogicaEscenario3* al tercero, y ambas tres tienen los mismos métodos.

Estas tres clases manejan cada una la matriz *mundo* rellenándola con valores distintos a través de su método *cargaMatriz*. Pero antes de todo esto, cuando se crea un objeto de

esas clases en función de la variable *numeroEscenario* de la clase *Proyecto*, *mundo[][][]*se inicializa en cada una de sus posiciones a "0", mediante el método *inicializaMatriz* que albergan. Con esto se obtendrían escenarios vacíos, como se ha visto con anterioridad, por lo que con el método anterior *cargaMatriz*, cambian en el array *mundo[][][]* los valores de determinadas coordenadas para conformar el escenario.

También es en estas clases donde se determina qué objetos tridimensionales se necesita para cada uno de los elementos que puede haber en el escenario en función del valor almacenado en *mundo[][][]*, qué textura se aplica a éstos y qué tamaño deben tener. Todo esto se realiza en el método *inicializaObjetos*, mediante llamadas a un método para guardarse en un vector de *características* perteneciente a la librería del PFC implementado. El orden en el que se van estableciendo las características es en función del valor que se corresponde en la matriz *mundo*, es decir, primero se guardarían las características del valor "1" que corresponde a un bloque del escenario, segundo las características de "2" que son los elementos pinchos…[103]

Asimismo mediante el método *inicializaFondo*, indicamos al motor gráfico qué objeto tridimensional queremos asociar al fondo, sus dimensiones y la imagen asociada.

Y finalmente el método *damePosición* indica qué número corresponde al objeto que queremos dibujar de más en la matriz *mundo*.

4.3.3.7 Carga

Se explica de forma resumida qué es lo que se produce durante la carga del entorno mencionado, dado que no incumbe a este proyecto, y de manera más amplia cómo se emplea ese tiempo de carga para mostrar una pantalla en la que se explica cómo jugar al videojuego.

En *Videojuego* se crea una distribución de la pantalla mediante código, al contrario que los demás casos que cargaban recursos *layout XML*. Se utiliza un *FrameLayout* que se caracteriza por que a medida que se añaden elementos a la pantalla, se van disponiendo éstos encima del anterior provocando que puedan tapar a los que se encuentren por debajo. Toda esta distribución conduce a la que se mostrará al usuario durante el desarrollo de la partida.

El primer elemento y el más importante que se agrega y que por lo tanto los demás le cubrirán, es un *GLView*, es decir, una vista personalizada donde se visualiza el escenario del juego, el personaje que el usuario controla y los enemigos que le rodeen. Su tamaño es de las mismas dimensiones que el del dispositivo móvil. Posteriormente, se incorporan más elementos que no se describen.

La carga de *GLView* necesita mucho tiempo porque prepara el entorno 3D del juego. Éste define su propia clase, la cual extiende de *SurfaceView*, que es un tipo especial de vista que se utiliza para 3D y se utiliza para cualquier vista que utilice *OpenGL*, como en este caso. *SurfaceView* utiliza a su vez una *Surface*, que es un lugar para el dibujo, como *Canvas*, excepto que es implementado por el hardware 3D. La superficie se establecerá cuando la *Activity Videojuego* esté funcionando en primer plano y cuando se produzca, creará su propio hilo para el dibujado de la pantalla. En la clase *GLView* se

declara un objeto hilo llamado *glThread* de la clase *Lógica*, pero que ésta hereda de *Dibujar* que es la que verdad extiende de *Thread*. Cuando la superficie *OpenGL* se ha creado, comienza el hilo *glThread* para manejar el dibujado de la pantalla, y cuando la superficie esté a punto de ser destruida, finalizará.

En referencia al presente proyecto, se explican los pasos que se realizan para la carga del entorno.

En la clase *Videojuego*:

1. Se inicializa el objeto de la clase *LogicaProtagonista* con las posiciones iniciales de éste sobre el escenario, uno de la clase *Cámara*, un objeto *Lógica* y la superficie de dibujado *GLView*.

En la clase *GLView*:

2. Cuando la *Surface* halla sido creada se inicia el hilo de *Dibujado*.

Posteriormente en la clase *Dibujar* y *Lógica*:

3. Se inicializa a "0" todos los valores de la matriz de enteros *mundo[][][]*, mediante el método *inicializaMatriz* de la clase *LogicaEscenarioX*.

4. La matriz anterior se carga de valores en determinadas posiciones que conformarán el escenario por donde debe avanzar el personaje, los objetos pincho, la llave que se debe de coger, la meta… Se realiza en las clases *LogicaEscenario1*, *LogicaEscenario2* o *LogicaEscenario3* en función de la variable *numeroEscenario* de *Proyecto*.

5. Se crea un array de *LogicaEnemigo* en función del número de enemigos, se generan sus posiciones iniciales y el tipo de *LogicaEnemigo* que son, de forma aleatoria mediante la llamada al método *dameAleatorios* de la clase *Lógica*, aunque también está preparado para poder colocar los enemigos en posiciones determinadas. Éste devuelve un array de números siendo los tres primeros las posiciones iniciales de éstos, comprendidos entre las dimensiones máximas y mínimas de cada eje de los escenarios. Estos deben cumplir que en esas posiciones debe de existir un "1" en la matriz del escenario correspondiente, que encima de esa posición no se encuentre un objeto pincho, es decir, que no valga "3" y que tampoco valga "1". La posición X de estos enemigos se establece de tal modo que no puedan estar muy cerca del usuario dado que enseguida se produciría su eliminación. Una vez conseguido, será la posición del *LogicaEnemigo* inicial. Y el último número es el correspondiente al tipo de *LogicaEnemigo* que debe ser.

6. Se habilita la variable *marcha* de la *Activity Videojuego* para dar por finalizado la carga de los elementos y de este modo cerrar la pantalla que genera el hilo de *Carga*.

Hasta que se produce este último paso, por ser el tiempo en que se carga el entorno y de la vista 3D amplio, se decidió en emplearlo para mostrar una ventana con información

de cómo jugar al videojuego. Para este cometido se crea un nuevo hilo junto con el de dibujado de *glThread*, hasta que éste último haya realizado la inicialización y carga de todo lo citado con anterioridad y pueda el usuario comenzar la partida. Por lo tanto, desde que el jugador presiona el botón *Jugar* del menú principal se crean esos dos hilos, y el principal seguirá con su ejecución normal, dado que si se finaliza, también lo hará la aplicación. Estos son los tres hilos existentes en la aplicación:

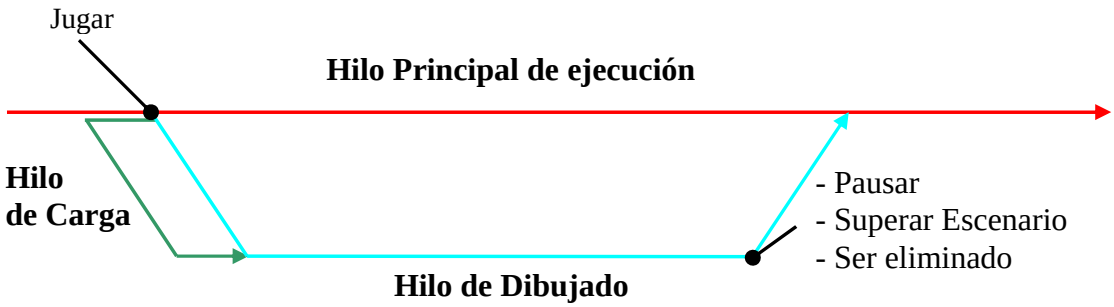


Figura 77. Hilos de ejecución en la aplicación

El hilo de *Dibujado* correspondería al hilo de *glThread* creado, que realiza de forma no visible al usuario la inicialización y carga de los objetos necesarios para poder comenzar una partida.

Durante estos procesos, el hilo de *Carga* muestra una ventana, debido a que un solo hilo no podría realizar ambas cosas. Esta ventana es un *Dialog* personalizado al estilo del videojuego, el cual carga su distribución mediante un recurso *layout XML* llamado *progress.xml*. Éste contiene elementos como el icono de la aplicación, el titulo del *Dialog*, un *ProgressBar* para dar la sensación de carga, un *TextView* y una *ImageView* que es la que muestra la información relacionada con el cómo se debe jugar a la aplicación.

Esta ventana es visible solo durante la carga, en el que el hilo se encuentra esperando a que el *boolean marcha* de la *Activity Proyecto*, cuyo significado es el estado en el que se encuentra la carga de los elementos del hilo de *Dibujado*, se encuentre habilitado para poder cerrarla. Cuando suceda, paso 6, saca el *Dialog* de la pantalla porque detrás de éste ya se podrá visualizar el entorno de desarrollo del juego.

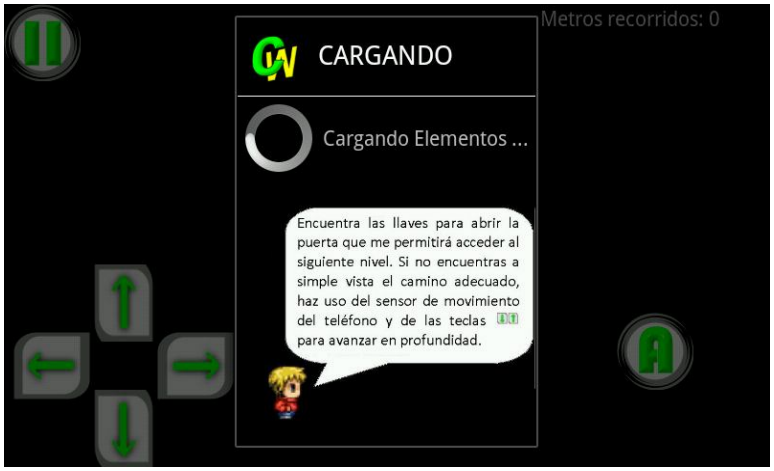


Figura 78. Captura de carga

Una vez llegados a este punto, el hilo *glThread* inicia el bucle principal de éste, donde se inicia la lógica principal del videojuego *Cube´s War*, siendo la parte fundamental para el presente proyecto.

4.3.3.8 Botones

Una vez eliminada la ventana de carga, el usuario se encuentra con la distribución de elementos que anteriormente se ha descrito. Algunos de esos botones que lo conforman, como los que constituyen el pad y el botón *Saltar*, poseen una característica particular que es la de no tener escuchador de eventos asociados. Esto es producido dos causas:

1. Al presionar éstos se producía un efecto de hundimiento del botón que no quedaba estético.

2. No se podía detectar la pulsación de dos botones a la vez y en la lógica del videojuego el personaje debe poder saltar y desplazarse al mismo tiempo.

Por lo tanto se optó por detectar los eventos que se producen sobre la superficie de dibujado del videojuego, es decir, sobre la *GLView*, para no ocasionar los dos hechos comentados anteriormente. Para ello en la clase *Videojuego* sobre el objeto *GLView* creado implementa el escuchador *onTouchEvent*, capturando los eventos necesarios para el movimiento del personaje previa comprobación de si los pixeles donde se producen el evento, coinciden con la posición de los botones en la pantalla, consiguiendo de este modo la simulación de la pulsación de éstos.

También es en este método donde se actualiza el *TextView* de la *Activity Videojuego* con la información de los metros recorridos. Estos metros no es más que la variables *posX* del protagonista sobre la matriz *mundo*.

Son muchos los eventos que se pueden producir sobre la pantalla táctil, pero son los que se describen a continuación los capturados por la aplicación:

- ***ACTION_DOWN*:** El usuario ha tocado con un dedo la pantalla. Este evento se produce en el mínimo instante en el que un solo dedo entre en contacto con la pantalla. Se utiliza para detectar las pulsaciones de los botones del pad superior e inferior, y de este modo mover al personaje en el eje –Z o +Z, respectivamente. Por lo que hasta que no se levante el dedo y se coloque de nuevo sobre la pantalla, no volverá a realizar esa acción determinada.

- ***ACTION_MOVE*:** El usuario realiza una pulsación dilatada en el tiempo o mueve el o los dedo/s que tiene en contacto con la pantalla, como máximo dos al estar *Android* 2.X limitado a ese número, aunque soporte en teoría hasta 256. Se usa sobre los botones del pad izquierdo y derecho, y sobre el botón *Saltar*. Sirve para mover al personaje en el eje -X y +X respectivamente, y que salte o no a la vez. Si se produce sobre los pixeles donde está situado el botón *Saltar*, se guardará una variable con la información del identificador del dedo que lo ha pulsado. Esto consigue que hasta que ese identificador del dedo no se ha levantado sobre la pantalla, no se pueda volver a realizar la acción de *Saltar*, logrando que solo se realice una vez la acción por cada pulsación. Esto simula

que el evento sea igual que el *ACTION_DOWN*, pero éste último no puede detectar más de un dedo sobre la pantalla, por lo que se desechó la idea.

- ***ACTION_UP*:** El usuario levanta el último o el único dedo que tenía sobre la pantalla. Se utiliza sobre los pixeles del botón *Saltar* para determinar si el dedo que lo había pulsado, se ha alzado de la pantalla. Se logra comparando el identificador del dedo que ha provocado el evento con el del caso de *ACTION_MOVE*. Si es así, se habilita de nuevo el poder saltar por parte del usuario.

- ***ACTION_POINTER_UP*:** El usuario levanta uno de los dos dedos que tiene sobre la pantalla. Al igual que el caso anterior, se usa para comprobar si el dedo que lo ha producido, además de ser sobre los pixeles donde se encuentra ubicado el botón *Saltar*, tiene el mismo identificador que en el del caso *ACTION_MOVE* y permite poder saltar de nuevo.

Por lo tanto, el único botón que queda es el de *Pausa* siendo el único que tiene asociado un escuchador de eventos de pulsación. Genera, cuando se hace clic sobre él, un *Dialog* adecuado a la estética del videojuego, obteniendo su distribución a partir del recurso *XML dialogo_pausa.xml*.

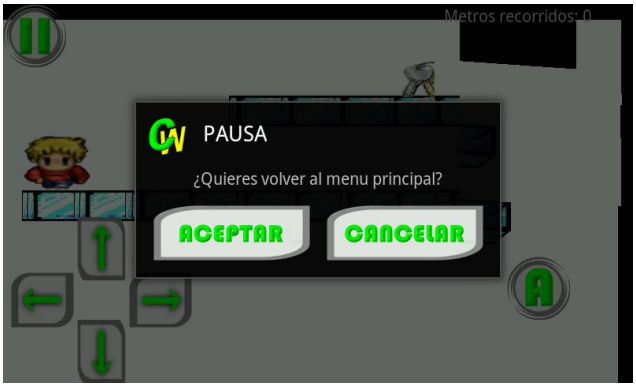


Figura 79. Juego pausado

En la ventana se muestra el icono de la aplicación, el titulo del *Dialog*, un *TextView*, y dos *Buttons*. El botón *Aceptar* retorna al menú principal realizando unas determinadas acciones, mientras que el botón *Cancelar* reanuda la partida, previo cierre del *Dialog* visualizado. Esta misma acción se consigue de igual modo pulsando el botón físico ↩ del dispositivo móvil, dado que el *Dialog* tiene asociado un escuchador de eventos de botones físicos.

4.3.3.9 Movimiento de la cámara asociado al sensor

Android es compatible con una amplia variedad de sensores que pueden ser utilizados por las aplicaciones para la captura de información del entorno del teléfono. Existen sensores como el acelerómetro, el giroscopio, campo magnético, de luz, de proximidad, de aceleración lineal, de orientación…

En la aplicación se utiliza este último, dado que nos proporciona información sobre cuánto de desplazado se encuentra el dispositivo móvil con respecto a sus ejes, mostrándolos continuación:

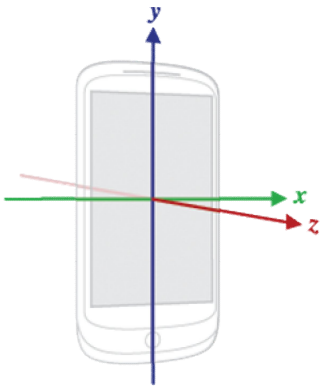


Figura 80. Sistema de coordenadas de *Android* [104]

Para ello se crea una instancia de *SensorManager* en la *Activity Videojuego*, con el fin de obtener información acerca de los sensores disponibles. No se necesita permisos para acceder al servicio del sensor. Una vez realizado, se registra a éste un escuchador de eventos de los sensores, *SensorEventListener*, que se halla también en la clase *Videojuego*. Es posible especificar la velocidad de entrega para los eventos del sensor, escogiendo para este proyecto, una velocidad adecuada para interfaces de usuario con la constante *SensorManager.SENSOR_DELAY_UI*. El escuchador implementa dos métodos asociados a éste, como son *onAccuracyChanged*, para cuando la exactitud del sensor ha cambiado, y *onSensorChanged*, llamado cuando los valores de los sensores cambian. Es en este último donde se establece que los valores que se quieren capturar son los del sensor tipo *Sensor.TYPE_ORIENTATION*.

Los valores que se capturan son recibidos en un array llamado *values[]* de dimensión 3, por lo que quiere decir que son tres los valores que se devuelven. Cada uno de estos representa ángulos expresados en grados cuyo significado es:

- **values[0]:** *Azimut*, ángulo entre la dirección del norte magnético y el eje Y, en todo el eje Z (0 a 359). 0 = Norte, 90 = Este, 180 = Sur, 270 = Oeste.

- **values[1]:** *Pitch*, rotación alrededor del eje X(-180 a 180), con valores positivos cuando se mueve el eje Z hacia el eje Y.

- **values[2]:** *Roll*, rotación alrededor del eje Y (-90 a 90), con valores positivos cuando se mueve el eje X hacia el eje Z.

El primer valor no interesa, porque dependiendo del lugar en el que se encuentre el dispositivo móvil relativo a la posición del norte magnético, varía su contenido. Los dos siguientes si, capturándose y mandándose a la clase *Cámara* donde serán tratados.

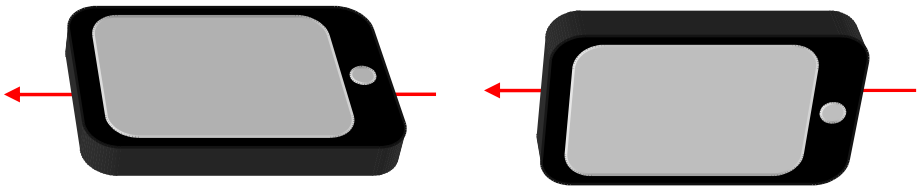


Figura 81. Movimientos del dispositivo móvil values[2]

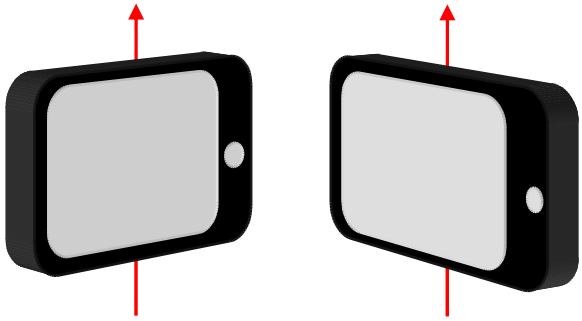


Figura 82. Movimientos del dispositivo móvil values[1]

Estos valores del sensor se utilizan para mover la cámara que enfoca al entorno tridimensional del videojuego con los movimientos que sufra el terminal, de tal modo que se hace rotar a ésta sobre la superficie de una esfera y luego que enfoque al personaje principal, es decir, en su posición en el eje X e Y. Consecuentemente, para conseguir el efecto 3D con el movimiento del dispositivo móvil lo que varía es la posición de la cámara con respecto al escenario, mientras lo que permanece fijo es el punto al que enfoca. Este punto varía en el caso de que el personaje se encuentre a la izquierda o derecha del nivel, por lo que su punto central de enfoque deja de ser el personaje por tener que desplazar la cámara un poco para enfocar más nivel.

Cabe mencionar que cuando se inicia una partida o un nivel, la cámara no enfoca directamente al personaje, sino al centro del escenario que se puede visualizar en ese momento. La cámara no centrará su punto de enfoque al personaje hasta que éste no haya llegado al centro de la pantalla, para posteriormente seguirle con su movimiento.

La obtención de los valores que proporciona el sensor de Orientación se produce cuando la *Activity Videojuego* está creada y en primer plano. Pero cuando ésta se encuentre pausada, en la implementación del método *onPause* de *Videojuego* se dejarán de registrar eventos de los sensores por lo que a la instancia del *SensorManager* se le retirará el *SensorEventListener*. Éste será de nuevo asignado cuando la *Activity* se reanude.

4.3.3.10 Lógica

Una vez cargados todos los componentes y entornos necesarios para poder comenzar una partida, *glThread* inicia el bucle principal. Éste se encuentra condicionado por el *boolean muerto* de la propia clase *Dibujar*, ejecutándose hasta que su valor sea *true.*

En Código 12 se mostraba como *Proyecto* iniciaba la nueva *Activity Videojuego* pero da tal forma que al provocarse la finalización de éste, debe devolver resultados. Esto es debido a que pueden ser distintas causas las que provoquen que *muerto* valga *true* con la

consecuente salida del bucle principal. Ese hecho producido por una de las distintas causas, no deben conducir a un mismo fin, que es el de volver a mostrar el menú principal que se encontraba a la espera. Por lo tanto, se debe saber el por qué del cierre de la *Activity*, y con ello mostrar ventanas con distinta información dependiendo del caso.

Con Código 12 consigue que a *Videojuego* se le asocie el código 1111 que le representa, pudiendo haber sido cualquier otro valor. Podía darse el caso de que existieran otras *Activities* hijas de la aplicación que interesara que devolvieran resultados al finalizarse. Esto provocaría la no distinción entre ellas, por lo que una manera de conseguir esto es mediante la asignación de este código a cada una de las *Activities* hijas. Cuando se produzca la finalización de alguna de ellas, mediante la implementación del siguiente método por parte de la *Activity* padre, puede determinarse cual ha sido mediante su código asignado, además de comprobarse qué resultado devuelve y qué datos adicionales manda la *Activity* hija:

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data)
```

Código 13: Método retorno

Este método lo implementa la *Activity* principal *Proyecto*, pudiendo determinar con él mediante la interpretación de los resultados, cuál ha sido la causa que ha llevado a finalizar *Videojuego* y actuar conforme a ésta.

Exceptuando el caso que el usuario pausé la partida y salga de ésta, las causas que se explican a continuación de finalización de la *Activity Proyecto*, muestran un *Dialog* personalizado, el cual sigue la estética de colores y fuentes del videojuego con distinta información y opciones.

- **El usuario completa un escenario para que el sistema proceda a cargar el siguiente:** es el método *Ganar* de la clase *Lógica* quien realiza las siguientes acciones: crea un *Intent* con un *boolean* llamado *"ganar"* puesto a *true* añadido como información extra, devuelve el resultado *RESULT_OK* más el *Intent* anteriormente creado a *Proyecto*, para posteriormente finalizar la *Activity Videojuego*.

  Proyecto una vez cerrada la *Activity Videojuego*, regresa a primer plano y por haber iniciado esta última de modo que se espera que devuelva resultados, actúa el método *onActivityResult* anteriormente mencionado. Para determinar que se ha finalizado *Videojuego* por la causa mencionada, se comprueba si el código recibido (*requestCode)* es el 1111, si el resultado (*resultCode*) es de tipo *RESULT_OK*, los datos que devuelve el *Intent* (*data*) es el boolean "*ganar*" siendo su valor *true* y si por último la variable global de *Proyecto numeroEscenario* es inferior a *numMaxEscenario-1*.

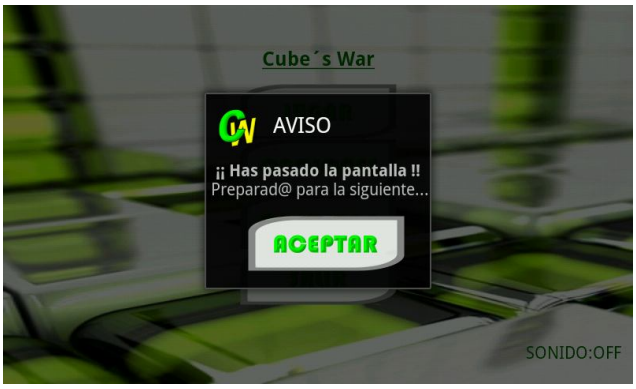


Figura 83. Completar escenario

Después de haber realizado todos los pasos anteriores e interpretado los datos que se devolvían, se muestra la ventana superior. Es un *Dialog* personalizado cuya distribución se obtiene a partir del recurso *XML dialogo_pantalla.xml*. Está formado por el icono de la aplicación, el titulo de la ventana, un *TextView* que obtiene su contenido a partir del fichero *strings.xml* y un *Button* cuya pulsación provoca que incremente la variable de *numeroEscenario*, cierre la ventana, produzca de nuevo la carga de todos los elementos y del siguiente escenario.

La ventana tiene asociado un escuchador de eventos de botones físicos, que mediante la pulsación de la tecla ⮌ del dispositivo móvil, permite al usuario si no quiere seguir con la partida, volver al menú principal, pero incrementándose de todas formas la variable *numeroEscenario* y de este modo no perder el avance conseguido.

- **El usuario supera el último escenario del videojuego por lo que ha finalizado la totalidad de éste:** se actúa en su totalidad de igual forma que en el anterior caso, excepto por la salvedad de que la última comprobación que se realiza es lo que la diferencia. En este caso, si la variable global *numeroEscenario* es igual a *numMaxEscenario-1* quiere decir que el usuario se encontraba en el último escenario y lo ha superado.

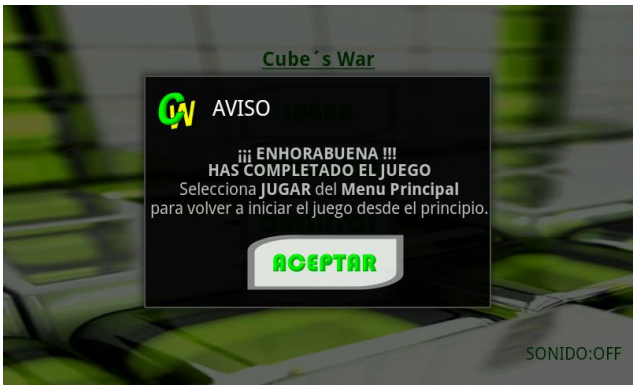


Figura 84. Completar juego

Si se ha culminado de forma acertada todas esas comprobaciones, se muestra un *Dialog* personalizado equivalente al caso anterior pero diferenciándose en que el texto que se muestra es referente a la causa que ha provocado la finalización de la *Activity Videojuego*. También, el botón mostrado realiza el cierre de la ventana, provocando que se vuelva a mostrar el menú principal, y el reinicio de

la variable *numeroEscenario*, del mismo modo que si se pulsa el botón físico ⮌ de la plataforma.

- **El usuario es eliminado del juego por producir que el personaje que maneja choque con los enemigos, con las piedras que descienden, caiga de la plataforma o sobre los objetos punzantes dispuestos en el escenario:** en este caso, es a través del método *Morir* de *Lógica* que crea el *Intent* como en los casos anteriores, añadiendo el mismo *boolean* "*ganar*" pero con su valor puesto a *false*.

  Una vez en el método *onActivityResult* de *Proyecto*, se realizan las comprobaciones y cuando se analice el contenido extra del *Intent* recibido se observará que su valor es *false*, por lo que se determina que se ha dado el caso de eliminación del personaje, mostrándose el siguiente *Dialog* cuya distribución es configurada a partir del recurso *XML dialogo_morir.xml*:

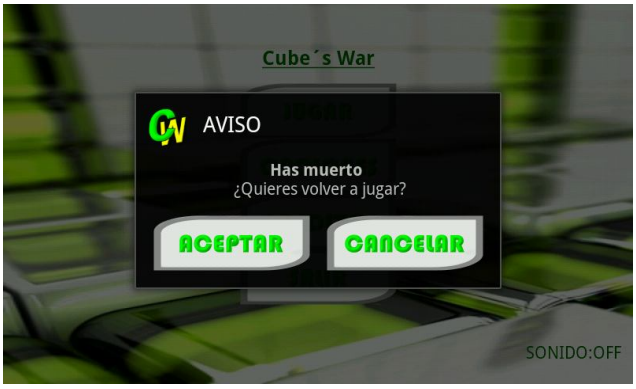


Figura 85. Jugador eliminado

Está formada por el icono de la aplicación, el titulo de la ventana, texto referente a la causa que provoca la finalización de la *Acitivity Proyecto* y dos botones. El botón *Aceptar* que se observa, inicia de nuevo la carga del escenario en el que se produjo la eliminación del usuario y cierra la ventana, mientras que el botón *Cancelar* o la pulsación del botón físico ⮌, cierra la ventana provocando que se muestre el menú principal de *Proyecto* que se encontraba a la espera.

- **BUCLE PRINCIPAL**

Una vez explicadas las condiciones de finalización del bucle principal de la aplicación y consecuentemente de la *Activity Videojuego*, se procede a exponer en orden los procesos que conforman el método *lógica* de la clase *Lógica* que es llamado sucesivamente en el bucle principal del videojuego.

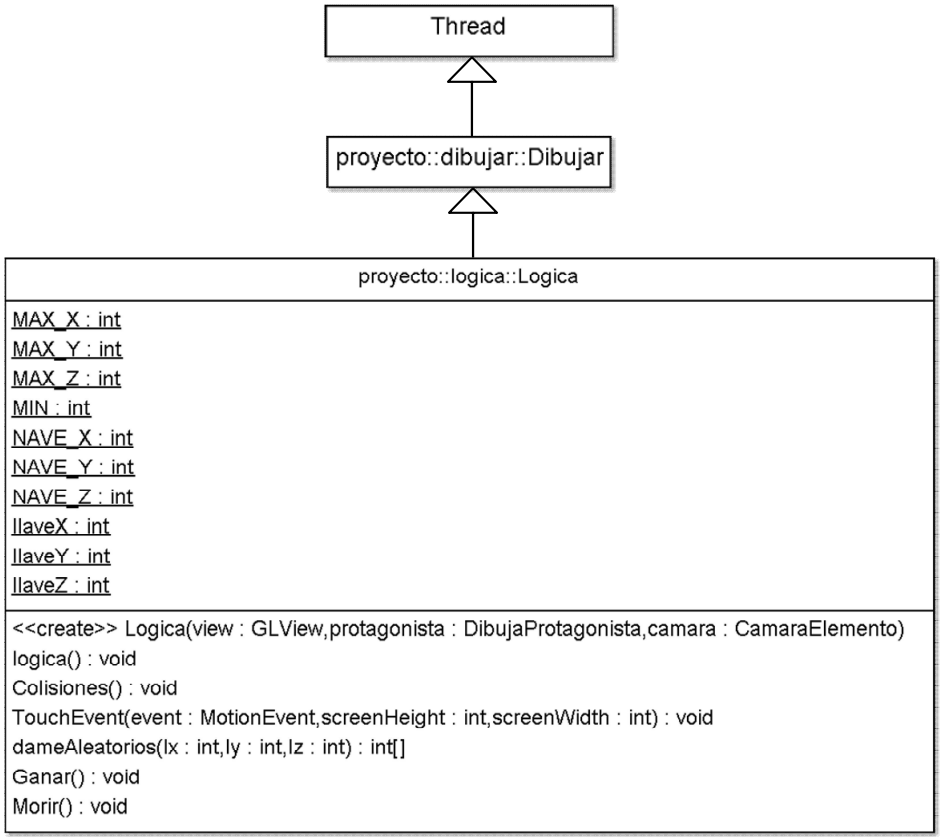


Figura 86. Clase *Lógica*

1. COLISIONES

La detección de colisiones es uno de los aspectos más importantes y necesarios de un videojuego. Sin estas comprobaciones un personaje podría atravesar una pared, un disparo no afectaría al enemigo o un campo de futbol no tendría límites.

Existen muchas y variadas formas de realizar la detección de colisiones, pero las condiciones en las que se ha desarrollado este juego, ha posibilitado en cierta manera el proceso.

El escenario, como se ha explicado anteriormente, se representa mediante el array tridimensional *mundo[][][]* anteriormente explicado. Cada uno de los valores guardados en él representa un elemento del escenario, pudiendo ser una plataforma, un objeto pincho, una llave o la meta del escenario.

Si se utiliza como índices las posiciones del protagonista, de los enemigos o de los meteoritos en la matriz *mundo[][][]*, determinando qué valor se tiene guardado en él, se sabrá si en la posición donde se encuentra o en su alrededor, existe plataforma o si se podrá avanzar en el sentido que se desee. El siguiente código es un ejemplo de cómo proceder de este modo:

```
if(Escenario.mundo[(int)(damePosX()+0.75f)][(int)damePosY()+3]
[(int)damePosZ()-1]!=1)
```

Código 14: Ejemplo colisiones

Se observa que a los distintos índices, que son las posiciones en los ejes, se suman o restan distintos factores. Esto es debido a que el objeto tridimensional que tienen asociados los enemigos y el protagonista, es un cubo plano de alto y ancho 1.5, al contrario que los cubos asociados al escenario que son de longitud 1, por lo que se tienen que realizar esos cálculos. Esos cubos tienen como referencia estándar la esquina inferior izquierda, como se muestra en la siguiente imagen:

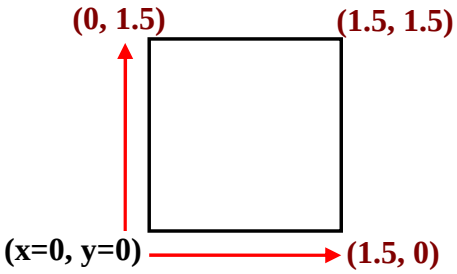


Figura 87. Cubo protagonista y enemigos

La posición X del protagonista, enemigos y meteoritos se tendrá como referencia el punto medio del cubo, es decir, su posición X más un factor de 0.75. Posteriormente además de esa cantidad se añade otra para determinar si en el sentido que se quiere avanzar lo permite el escenario. En la posición Y, dependiendo del caso que se quiera tratar, no se sumará factor alguno, se aumentará 1.5 o un factor mediante la realización de cálculos, como por ejemplo cuánto salta el personaje. Y finalmente en el Z se efectuará una suma o resta en 1 factor dependiendo del sentido de avance en ese eje.

La detección de colisiones implica la llamada de distintos métodos sucesivamente, hasta que se dé el primer método que se explica, provocando la rotura del bucle principal:

1. **Método *Colisiones* de la clase *Lógica*:** primero se comprueba si la posición Y del protagonista es inferior o igual a -0.6, dado que no se encontraría dentro de los limites establecidos en el escenario. Su significado es el de que el personaje se ha caído de una plataforma y ya no existe ninguna sobre la que apoyarse. Si se produce, el *boolean muerto* que permite la ejecución del bucle principal, pasa a tomar valor *true*, y se realiza una llamada al método *Morir*, explicado con anterioridad

   Posteriormente, se comprueba del mismo modo que en Código 14, si en la posición del personaje o debajo de éste en la matriz *mundo* hay un valor "2", es decir, un objeto pincho. Si se cumple, se actúa del mismo modo que en el caso anterior.

   A continuación, en lugar de que en la posición del personaje sobre el array *mundo* se verifique si existe un "2", se hace si el valor guardado es un "3", la meta. Si se formaliza y además la variable *LogicaProtagonista.llave* es *true*, es decir, que se ha recogido la llave del escenario, se realiza una llamada al método *Ganar* cambiando la variable global *muerto* a *true*, rompiendo el bucle principal, pudiendo avanzar al siguiente escenario o encontrarse en el caso de haber superado la totalidad del juego, explicado con anterioridad.

Finalmente se comprueba si son iguales cada una de sus posiciones en los ejes del array de enemigos y meteoritos con las del protagonista, para que de este modo actúe el método *Morir*.

2. **Método *Colisiones* de la clase *LogicaProtagonista*:** después de comprobarse que ninguna de las colisiones anteriores que pueden provocar la finalización del bucle principal y por lo tanto la finalización de la *Activity Videojuego*, se ha verificado, se procede a la comprobación de si hacia donde quiere avanzar el usuario es posible.

   Anteriormente a éste, el usuario ha pulsado algunos de los píxeles de la pantalla táctil bajo los que se encuentran los botones del pad y *Saltar*, por lo que se habilita la condición que informa que el usuario ha querido pulsar sobre un determinado botón y que se quiere avanzar en el sentido asociado.

   Entonces este método *Colisiones*, comprueba uno a uno esas condiciones. Dependiendo de cuál se haya habilitado, confirma si la posición en el eje del protagonista se encuentra dentro de los límites establecidos del escenario para posteriormente comprobar si hacia donde se quiere avanzar no existe un "1" en la matriz *mundo* mediante el modo Código 14, para permitir o no el avance. Si bajo la posición hacia donde se quiere avanzar no hay plataforma, depende del usuario no progresar en esa dirección.

   Si la posición del personaje principal coincide con la de la llave que se debe de recoger para completar los escenarios, modificará a la matriz *mundo* cambiando su valor almacenado en esa posición por un "0".

3. **Método *Comprobar* y *Colisiones* de la clase *LogicaEnemigo*:** se recorre el array de *LogicaEnemigos* para proceder a las comprobaciones. Primero se realiza la llamada a *Comprobar* que recibe la posición X y Z del protagonista para determinar en qué sentido tiene que avanzar para acercarse a él, es decir, si la posición X del enemigo es inferior a la posición X del protagonista, tendrá que avanzar hacia la derecha, si es inferior a la izquierda. El mismo caso sucede con la posición Z pero solo en las *LogicaEnemigo* de tipo 2, si es igual no tendrá que avanzar, si es inferior o mayor, progresará o retrocederá respectivamente. Por lo que se habilitan ciertas variables para determinar en qué sentido de los ejes tiene que avanzar.

   Y en segundo lugar se realiza la llamada a *Colisiones*, que actúa de forma idéntica que el método de *LogicaProtagonista*, pero si se realiza la comprobación de que hacia donde se quiere avanzar habrá plataforma sobre apoyarse, para de este modo progresar o no. Esto refleja inteligencia en los enemigos.

2. MOVER

Posterior a la comprobación de las colisiones por parte del protagonista y de los enemigos del juego, se procede a mover su posición. Es en el método *Mover* de las clases *LogicaProtagonista* y *LogicaEnemigo* que realiza éste cometido. Primero se verifica uno a uno qué condiciones ha habilitado el método *Colisiones* anterior, y

dependiendo de cual, se procede a incrementar las variables copias *dX*, *dY* y *dZ*, de las posiciones en los ejes de los elementos, una cantidad determinada por las variables *velocidadX*, *salto*, *velocidadZ* y para el protagonista, o *desplazamientoX* y *desplazamientoZ* para los enemigos, dado que ellos no pueden variar su posición Y por encontrarse siempre a la misma altura.

También se realiza la llamada al método *Mover* de la clase *Cámara*. Anterior a éste, el sensor de Orientación ha llamado al método *Actualizar* pasando sus valores e igualando a éstos las variables copias *dY* y *dZ*. *Mover* los trata estableciendo unos valores máximos y mínimos a cada uno de ellos, produciendo si son superados, la igualación a esos límites y posteriormente la igualación de *dY* y *dZ* a *posY* y *posZ*.

3. ACTUALIZAR

Una vez desplazada la posición sobre el escenario del protagonista y de los enemigos, solo queda copiar los valores de las variables *dX*, *dY* y *dZ* a las de *posX*, *posY* y *posZ*, siendo los métodos *Actualizar* de las clases *LogicaProtagonista* y *LogicaEnemigo* quienes realizan este cometido.

Igualmente se produce la llamada al método *actualizaMeteorito* de la clase *LogicaMeteorito*. En él se produce todas los posibles cálculos de sus movimientos dado que al ser escasos, no se necesita tiempo alguno en su ejecución. Además, si se produce alguna colisión del meteorito con el escenario, este se destruye en el punto en el que confluyan, es decir, se utiliza como índices su posición en los ejes dentro de la matriz *mundo[][][]* y sustituye esa variable guardada a "0". También en él se modifica la posición en el eje Y de los elementos meteoritos dispuestos sobre el escenario.

Capítulo 5

Conclusiones

5.1 Conclusiones

Tras haber concluido el desarrollo de este proyecto fin de carrera, es el momento de efectuar análisis y balance del resultado final obtenido, recordando todo el camino andado, siendo la manera más idónea para poder obtener conclusiones. Éstas comprenden distintos aspectos, desde el técnico y profesional con el desarrollo del videojuego, como en el ámbito personal por ser un proyecto que implica el uso de otro, en el que se debe aprender a trabajar en equipo.

Con este proyecto se ha conseguido ampliar mis conocimientos que tenía sobre este sistema operativo. Ampliarlos, debido a que anteriormente en una asignatura de la carrera me había formado en él y había desarrollado aplicaciones pero de otra índole. El proyecto ha supuesto dar un paso más en esos conocimientos, y que gracias a la amplia documentación que *Google* pone a disposición del desarrollador, la cantidad de foros que existen para asistir al usuario y la ayuda que siempre tenía del tutor, han conseguido que pudiera ser posible. También aprender más sobre el desarrollo de los videojuegos, campo que desconocía en cierta medida pero en el cual tenía mucha ganas de introducirme, y que mediante documentación existente en distintas fuentes, he podido conseguir. Aprender sobre el ciclo de vida de un videojuego, manejar sus estados,

personajes, enemigos, escenarios, movimientos, las colisiones… han sido aspectos muy interesantes de formarse.

En otro ámbito, el haber cursado una carrera en el que el principal lenguaje de programación que se nos ha enseñado ha sido *Java*, suponía poder hacer uso de muchos de los conocimientos adquiridos durante la etapa universitaria, y de este modo ver reflejados todos ellos en un proyecto final y sentirte recompensado en cierta manera.

Desde un punto de vista personal, la realización de este proyecto me ha servido para descubrir un gran interés por el desarrollo de aplicaciones móviles de *Android*, dado que considero que la estrategia que está siguiendo y que a su vez sea código abierto, ofrece un gran abanico de posibilidades, teniendo ante mí una muy buena oportunidad de futuro.

Dentro de este ámbito, el haber sido un proyecto que implica la utilización de una biblioteca desarrollada como PFC de una compañera, evolucionando paralelamente con el mío, para que su unión dé como fruto final un videojuego de plataformas, conlleva trabajo de equipo. Por lo tanto, no ha sido algo habitual dentro de los PFCs porque implica la participación de dos alumnos hacía un mismo objetivo, por lo que no existe una única forma de actuar, si no dos. Dos formas de pensar y ver las cosas diferentes, las cuales tenían que llegar a un punto común para conseguir el desarrollo del videojuego. Esto conforma una dificultad, que mediante el entendimiento, la comunicación, la amistad que existía anteriormente, y poniendo cada uno de su parte, se ha llegado a conseguir con balance positivo.

En el desarrollo de cada uno de nuestros proyectos, que suponen los módulos de lógica e interfaz del videojuego, no ha existido problema, solo los posibles obstáculos encontrados a la hora de poder conseguir ciertos objetivos. Pero a la hora de aunar todo esos trabajos en el videojuego final, si que ha habido problemas. Obviamente todo se tiene que conseguir mediante trabajo y comunicación entre ambas partes, y siempre hay pequeños problemas que surgen o por mal entendimiento de las directrices o por otros aspectos, pero todo tiene posible solución, y más en la programación, que se puede rehacer las veces que se quiera. Pero todo este trabajo en equipo desarrollado, de forma inconsciente, nos preparó en cierta medida para un futuro profesional cercano, en el que este factor se dará cada día.

El resultado obtenido en el videojuego final considero que es notable, teniendo en cuenta que dos personas que no se han formado en su totalidad en conocimientos informáticos, hayan conseguido desarrollar un videojuego para dispositivos móviles y en un entorno de 3D. Cierto es, que existen puntos que podrían mejorarse sensiblemente al no ser experimentados en la materia correspondiente, pero aún así, el balance final es muy positivo.

Capítulo 6

Trabajos futuros

6.1 Trabajos futuros

En este proyecto de fin de carrera se ha creado un videojuego para el sistema operativo *Android*, rompiendo con lo establecido en los juegos de plataformas, dado que además de existir la posibilidad de avanzar en la pantalla o retroceder, también se puede mover uno en la profundidad del escenario. Este aspecto se podría implantar en muchos de los videojuegos ya establecidos y conocidos en el mercado o para los nuevos de este mismo tipo, dado que aumentarían los contenidos de ellos en tal medida que daría lugar a mayores posibilidades en los escenarios, pudiendo idear nuevas estrategias e invenciones en este tipo de videojuegos.

Para poder extenderse estas características a los demás videojuegos, en la plataforma en el que se desarrolla, tendría que contener esos sensores que me han brindado la posibilidad de haber desarrollado esta idea, ya que con simples movimientos del dispositivo móvil, se han obtenido los datos que ofrecen los sensores y se han aportado al videojuego. Además se tendría que convertir el entorno donde se desarrolla el videojuego del que se quiere extender la lógica desarrollada en el presente proyecto, a uno en 3D.

El desarrollo completo del videojuego se ha hecho con productos de software libre, como es Eclipse, además del plugin que se aplica a éste para desarrollar aplicaciones *Android*, por lo que su implementación por parte de otras personas que le interese este proyecto es razonable. Por lo tanto, algunas características o actividades del proyecto se pueden extender incluso llegar a mejorar.

Esas extensiones que se podrían realizar sobre el videojuego desarrollado, serían algunas de las ideas que se exponen a continuación, que son:

- Ampliación de la longitud de los escenarios que se presentan y/o el número de ellos, para que de esta manera no se pueda quedar corta la posible duración y experiencia del videojuego.

- Considerar la posibilidad de implementar que el protagonista no estuviera asociado a un cubo plano al que se le inserta una imagen *Sprite*, si no la de crear su modelo en 3D. Esto también se puede aplicar de manera análoga a los enemigos que nos encontramos durante la ejecución del videojuego, y que por falta de tiempo y considerable complejidad, no se pudo llevar a cabo. Por otra parte y relacionado con los personajes, también se podría brindar al usuario la elección de su personaje entre unos determinados, para que fuese un videojuego más personalizado y de alguna manera, se pueda sentir identificado con el personaje que controla.

- Ofrecer la posibilidad de la interconexión de dos móviles mediante la tecnología *Bluetooth*[105] o conexión a Internet tanto por *Wifi*[106] o por la red móvil, para que ambos dispositivos pudieran jugar, y de esta manera ofrecer un modo de juego más como es el de multijugador.

- Elegir el nivel desde donde se quiere comenzar. Esta posibilidad solo se mostraría cuando el usuario hubiera completado con éxito todos los niveles que hay en el videojuego.

- Poder ampliar a otras plataformas, como las tablets[107], el desarrollo de este videojuego consiguiendo su expansión, y de esta manera, otros usuarios puedan entretenerse con éste.

Bibliografía y referencias

[1] Intel, “Intel”, http://www.intel.com/content/www/us/en/homepage.html [Último acceso Agosto 2011]

[2] Pbs, “Gordon Moore”, http://www.pbs.org/transistor/album1/moore/index.html [Ultimo acceso Agosto 2011]

[3] Intel, “Ley de Moore”, http://www.intel.com/cd/corporate/techtrends/emea/spa/209840.htm [Ultimo acceso Agosto 2011]

[4] Sgm, “II Guerra Mundial”, http://sgm.casposidad.com/ [Ultimo acceso Agosto 2011]

[5] Cellphones, “Smartphones”, http://cellphones.about.com/od/smartphonebasics/a/what_is_smart.htm [Ultimo acceso Agosto 2011]

[6] Garmin, “GPS”, http://www8.garmin.com/aboutGPS/ [Ultimo acceso Septiembre 2011]

[7] Masadelante, “Sistema operativo”, http://www.masadelante.com/faqs/sistema-operativo [Ultimo acceso Septiembre 2011]

[8] Expansion, “Ventas smartphones”, http://www.expansion.com/2011/02/09/empresas/digitech/1297213767.html [Ultimo acceso Agosto 2011]

[9] Apple, “iOS”, http://www.apple.com/ios/ [Ultimo acceso Septiembre 2011]

[10] Symbianos, “Symbian OS”, http://www.symbianos.org/intro [Ultimo acceso Septiembre 2011]

[11] Android, “Android”, http://www.android.com/ [Ultimo acceso Septiembre 2011]

[12] Gartner, “Ventas S.O.”, http://www.gartner.com/it/page.jsp?id=1689814 [Ultimo acceso Agosto 2011]

[13] Open Handset Alliance, “Open Handset Alliance”, http://www.openhandsetalliance.com/ [Ultimo acceso Agosto 2011]

[14] Google, “Google”, http://www.google.com/intl/en/about/corporate/index.html [Ultimo acceso Septiembre 2011]

[15] Apache, “Apache”, http://www.apache.org/ [Ultimo acceso Septiembre 2011]

[16] Debian, “Linux”, http://www.debian.org/ [Ultimo acceso Septiembre 2011]

[17] Java, “Java”, http://java.com/es/about/ [Ultimo acceso Septiembre 2011]

[18] Dalvikvm, “Dalvik”, http://www.dalvikvm.com/ [Ultimo acceso Septiembre 2011]

[19] Khronos, “OpenGL ES”, http://www.khronos.org/opengles/ [Ultimo acceso Septiembre 2011]

[20] Android, “SDK”, http://developer.android.com/sdk/index.html [Ultimo acceso Agosto 2011]

[21] Android, “API”, http://developer.android.com/guide/appendix/api-levels.html [Ultimo acceso Septiembre 2011]

[22] Android, “Android Market”, https://market.android.com/?hl=en [Ultimo acceso Septiembre 2011]

[23] W3, “XML”, http://www.w3.org/XML/ [Ultimo acceso Septiembre 2011]

[24] Eclipse, “Eclipse”, http://www.eclipse.org/org/ [Ultimo acceso Septiembre 2011]

[25] Alt1040, “OXO”, http://alt1040.com/2011/07/oxo-un-videojuego-para-uno-de-los-primeros-computadores-de-la-historia [Ultimo acceso Septiembre 2011]

[26] Cam, "Univesidad de Cambridge", http://www.cam.ac.uk/ [Ultimo acceso Septiembre 2011]

[27] Vida extra, "William Higinbotham", http://www.vidaextra.com/industria/el-padre-de-los-videojuegos-william-higinbotham-su-historia-su-leyenda [Ultimo acceso Septiembre 2011]

[28] Bnl, "Tennis for two", http://www.bnl.gov/bnlweb/history/higinbotham.asp [Ultimo acceso Septiembre 2011]

[29] Pong Story, "Pong", http://www.pong-story.com/intro.htm [Ultimo acceso Septiembre 2011]

[30] Inventors, "Spacewar", http://inventors.about.com/od/sstartinventions/a/Spacewar.htm [Ultimo acceso Septiembre 2011]

[31] Computer Space fan, "Computer Space", http://www.computerspacefan.com/ [Ultimo acceso Septiembre 2011]

[32] Inc, "Nolan Bushnell", http://www.inc.com/magazine/20090401/the-gamer.html [Ultimo acceso Septiembre 2011]

[33] Atari, "Atari", http://www.atari.com/ [Ultimo acceso Septiembre 2011]

[34] Ralph Baer, "Ralph Baer", http://www.ralphbaer.com/ [Ultimo acceso Septiembre 2011]

[35] Magnavox Odyssey, "Magnavox Odyssey", http://www.magnavox-odyssey.com/ [Ultimo acceo Septiembre 2011]

[36] Empire Arcadia, "Ted Dabney", http://www.empirearcadia.com/legends/p0009.htm [Ultimo acceso Septiembre 2011]

[37] Atari museum, "Home Pong", http://www.atarimuseum.com/videogames/dedicated/homepong.html [Ultimo acceso Septiembre 2011]

[38] Skool days, "Indy 800", http://www.skooldays.com/categories/arcade/ag1177.htm [Ultimo acceso Septiembre 2011]

[39] All about Steve Jobs, "Steve Jobs", http://allaboutstevejobs.com/ [Ultimo acceso Septiembre 2011]

[40] Apple, "Apple Computer", http://www.apple.com/ [Ultimo acceso Septiembre 2011]

[41] Atariage, "VCS/2600", http://www.atariage.com/2600/index.html?SystemID=2600 [Ultimo acceso Septiembre 2011]

[42] Taito, "Taito", http://www.taito.com/ [Ultimo acceso Septiembre 2011]

[43] Star Wars, "Star Wars", http://www.starwars.com/ [Ultimo acceso Septiembre 2011]

[44] Spy hunter, "Asteroids", http://spyhunter007.com/silco_west.htm [Ultimo acceso Septiembre 2011]

[45] Free Pac Man, "Pac Man", http://www.freepacman.org/ [Ultimo acceso Septiembre 2011]

[46] Namco Bandai Games, "Namco", http://www.namcobandaigames.com/ [Ultimo acceso Septiembre 2011]

[47] Nintendo, "Nintendo", http://www.nintendo.com/countryselector [Ultimo acceso Septiembre 2011]

[48] Game and Wathc, "Game & Watch", http://www.gameandwatch.com/ [Ultimo acceso Septiembre 2011]

[49] Cyber Iapc, "Game Boy", http://www.cyberiapc.com/vgg/nintendo_gameboy.htm [Ultimo acceso Septiembre 2011]

[50] Classic games arcade, "Donkey Kong", http://www.classicgamesarcade.com/game/21595/Donkey-Kong-Classic-Game.html [Ultimo acceso Septiembre 2011]

[51] Nintendo, "Mario Bros", http://mario.nintendo.com/ [Ultimo acceso Septiembre 2011]

[52] Classic gaming, "Nintendo Entertainment System", http://classicgaming.gamespy.com/View.php?view=ConsoleMuseum.Detail&id=26 [Ultimo acceso Septiembre 2011]

[53] Tv tropes, "Bomber Man", http://tvtropes.org/pmwiki/pmwiki.php/Main/Bomberman [Ultimo acceso Septiembre 2011]

[54] Tetris, "Tetris", http://www.tetris.com/ [Ultimo acceso Septiembre 2011]

[55] Sega, "Sega", http://www.sega.es/ [Ultimo acceso Septiembre 2011]

[56] Old Computers, "Amiga 500", http://www.old-computers.com/museum/computer.asp?c=65 [Ultimo acceso Septiembre 2011]

[57] Wikia, "Mega Drive", http://castlevania.wikia.com/wiki/Sega_Mega_Drive [Ultimo acceso Septiembre 2011]

[58] Soni, "Soni the Hedgedog", http://www.sonicthehedgehog.com/ [Ultimo acceso Septiembre 2011]

[59] Vg Museum, "Game Gear", http://www.vgmuseum.com/systems/gg/ [Ultimo acceso Septiembre 2011]

[60] Old Computers, "Master System 2", http://www.old-computers.com/museum/computer.asp?c=1252&st=2 [Ultimo acceso Septiembre 2011]

[61] Neo Geo, "Neo Geo", http://www.neo-geo.com/ [Ultimo acceso Septiembre 2011]

[62] Sega Saturn, "Sega Saturn", http://www.sega-saturn.com/ [Ultimo acceso Septiembre 2011]

[63] Play Station, "Play Station", http://us.playstation.com/ [Ultimo acceso Septiembre 2011]

[64] Nokia, "Nokia", http://www.nokia.com/global/wayfinder [Ultimo acceso Septiembre 2011]

[65] Cool Buster, "Snake", http://www.coolbuster.net/2010/08/snake-game-on-youtube.html [Ultimo acceso Septiembre 2011]

[66] Dreamcast, "Dreamcast ", http://www.dreamcast-scene.com/ [Ultimo acceso Septiembre 2011]

[67] Xbox, "Xbox", http://www.xbox.com/es-ES/ [Ultimo acceso Septiembre 2011]

[68] Microsoft, "Microsoft", http://www.microsoft.com/en-us/default.aspx [Ultimo acceso Septiembre 2011]

[69] Halo, "Halo", http://halo.xbox.com/en-us [Ultimo acceso Septiembre 2011]

[70] Tcovg, "Ngage", http://tcovg.aftervision.com/index.php?page=ngage [Ultimo acceso Septiembre 2011]

[71] The Sims, "The Sims", http://thesims.ea.com/en_us/home [Ultimo acceso Septiembre 2011]

[72] Gameloft, "Gameloft", http://www.gameloft.es/ [Ultimo acceso Septiembre 2011]

[73] Angry Birds, "Angry Birds", http://chrome.angrybirds.com/ [Ultimo acceso Septiembre 2011]

[74] IDC, "Descargas apps año", http://www.idc.com/getdoc.jsp?containerId=prUS22917111 [Ultimo acceso Agosto 2011]

[75] Androlib, "Nuevas aplicaciones", http://es.androlib.com/appstats.aspx [Ultimo acceso Septiembre 2011]

[76] Chomp, "Datos apps favoritas", http://chomp.com/etc/chomp-charts/aug-2011 [Ultimo acceso Agosto 2011]

[77] Epic Games, "Epic Games", http://www.epicgames.com/ [Ultimo acceso Septiembre 2011]

[78] Apple, "AppStore", http://www.apple.com/mac/app-store/ [Ultimo acceso Septiembre 2011]

[79] Chomp, "Apps gratuitas vs de pago" http://chomp.com/etc/chomp-charts/aug-2011 [Ultimo acceso Septiembre 2011]

[80] Oled info, "Super amoled", http://www.oled-info.com/super-amoled [Ultimo acceso Septiembre 2011]

[81] Galaxy S2, "Galaxy S2", http://www.samsung.com/global/microsite/galaxys2/html/ [Ultimo acceso Septiembre 2011]

[82] Android, "NDK", http://developer.android.com/sdk/ndk/index.html [Ultimo acceso Septiembre 2011]

[83] Havok, "Havok", http://www.havok.com/ [Ultimo acceso Septiembre 2011]

[84] Unity, "Unity", http://unity3d.com/ [Ultimo acceso Septiembre 2011]

[85] Sony Ericsson, "Xperia Play", http://www.sonyericsson.com/cws/products/mobilephones/overview/xperia-play?cc=gb&lc=en [Ultimo acceso Septiembre 2011]

[86] Unreal Engine, "Unreal Engine", http://www.unrealengine.com/ [Ultimo acceso Septiembre 2011]

[87] Nvidia, "Nvidia", http://www.nvidia.com/content/global/global.php [Ultimo acceso Septiembre 2011]

[88] EA, "EA", http://www.ea.com/ [Ultimo acceso Septiembre 2011]

[89] Codemasters, "Codemasters", http://www.facebook.com/codemasters [Ultimo acceso Septiembre 2011]

[90] PlayStation Suite, "PlayStation Suite", http://uk.playstation.com/playstationsuite/ [Ultimo acceso Septiembre 2011]

[91] PlayStation Store, "PlayStationStore", http://us.playstation.com/psn/playstation-store/ [Ultimo acceso Septiembre 2011]

[92] Android, "Información OpenGL ES", http://developer.android.com/guide/topics/graphics/opengl.html [Ultimo acceso Septiembre 2011]

[93] Android, "Moviles versiones OpenGL ES", http://developer.android.com/resources/dashboard/opengl.html [Ultimo acceso Agosto 2011]

[94] Android, "Android manifest", http://developer.android.com/guide/topics/manifest/manifest-intro.html [Ultimo acceso Septiembre 2011]

[95] Android, "Activity", http://developer.android.com/reference/android/app/Activity.html [Ultimo acceso Septiembre 2011]

[96] Android, "Intent", http://developer.android.com/reference/android/content/Intent.html [Ultimo acceso Septiembre 2011]

[97] Android, "Ciclo vida Activity", http://developer.android.com/reference/android/app/Activity.html [Ultimo acceso Septiembre 2011]

[98] Android, "ADT", http://developer.android.com/sdk/eclipse-adt.html [Ultimo acceso Septiembre 2011]

[99] Android, "Formatos multimedia", http://developer.android.com/guide/appendix/media-formats.html [Ultimo acceso Agosto 2011]

[100] Android, "Android View", http://developer.android.com/reference/android/view/package-summary.html [Ultimo acceso Agosto 2011]

[101] W3 schools, "HTML", http://www.w3schools.com/html/default.asp [Ultimo acceso Septiembre 2011]

[102] Física práctica, "Tiro Vertical", http://www.fisicapractica.com/tiro-vertical-caida-libre.php [Ultimo acceso Agosto 2011]

[103] Proyecto Fin de Carrera, "Desarrollo de un motor grafico 3D para videojuegos de plataformas en Android", Cristina Mingueza Balsalobre 2011.

[104] Android, "Sensor Orientación", http://developer.android.com/reference/android/hardware/SensorEvent.html#values [Ultimo acceso Agosto 2011]

[105] Bluetooth, "Bluetooth", http://www.bluetooth.com/Pages/Bluetooth-Home.aspx [Ultimo acceso Septiembre 2011]

[106] WebOpedia, "Wifi", http://www.webopedia.com/TERM/W/Wi_Fi.html [Ultimo acceso Septiembre 2011]

[107] Mashable, "Tablet", http://mashable.com/follow/topics/tablets/ [Ultimo acceso Septiembre 2011]

[108] Kioskea, "Diagrama de Gantt", http://es.kioskea.net/contents/projet/gantt.php3 [Ultimo acceso Septiembre 2011]

[109] Youtube, "Youtube", http://www.youtube.com/t/about_youtube [Ultimo acceso Septiembre 2011]

[110] Tiempos modernos, "Ciclo de vida del producto", http://www.tiemposmodernos.eu/agcpe-marketing-mix/ [Ultimo acceso Septiembre 2011]

Anexo A

Planificación

Este anexo despliega la planificación inicial tanto del videojuego como del proyecto desarrollado, así como de la final realizada.

A.1 Planificación inicial del videojuego

En este apartado se incluye la planificación inicial realizada antes de comenzar con el desarrollo del proyecto. Se me adjudicó el 11 de Octubre de 2010, estimando que podría tenerlo acabado a finales de Febrero de 2011, habiendo realizado un trabajo total de unos 5 meses.

En primer lugar se especifican en orden cronológico las actividades que se llevarían a cabo para la realización del proyecto, incluyendo las horas que serían necesarias para llegar a desarrollar cada una de ellas, incluyendo una breve explicación:

HITOS	HORAS
Documentación inicial: informarse sobre todo lo que conlleva el sistema operativo *Android*.	48
Instalación del entorno: descargar e instalar todas las herramientas y programas necesarios para desarrollar aplicaciones sobre *Android*.	6
Familiarización con la plataforma: aprender y estudiar las posibilidades que ofrece el entorno de desarrollo instalado.	40
Análisis y diseño: realizar distintas propuestas y estudios preliminares del proyecto, para posteriormente diseñar y elegir las adecuadas.	24
Implementación: desarrollo de la lógica del videojuego planteado.	372
Integración: ligar la lógica del videojuego conseguida con la biblioteca desarrollada por otro PFC implementada por el proyecto.	40
Memoria: desarrollo de la memoria sobre el proyecto.	150
TOTALES	680 horas

Tabla 50: Planificación inicial del videojuego

Se debe tener en cuenta que el horario de trabajo que se emplearía para el desarrollo de la aplicación y memoria del proyecto sería de 8 horas diarias en días lectivos, por lo que no se incluyen los sábados y domingos.

A.1.1 Diagrama de Gantt

El diagrama de *Gantt*[108] sitúa de forma secuencial las diferentes etapas del proyecto a lo largo de una línea temporal que representa el tiempo total para la consecución del videojuego. En la siguiente gráfica refleja la planificación de éste, así como la fecha estimada para cada una de las tareas.

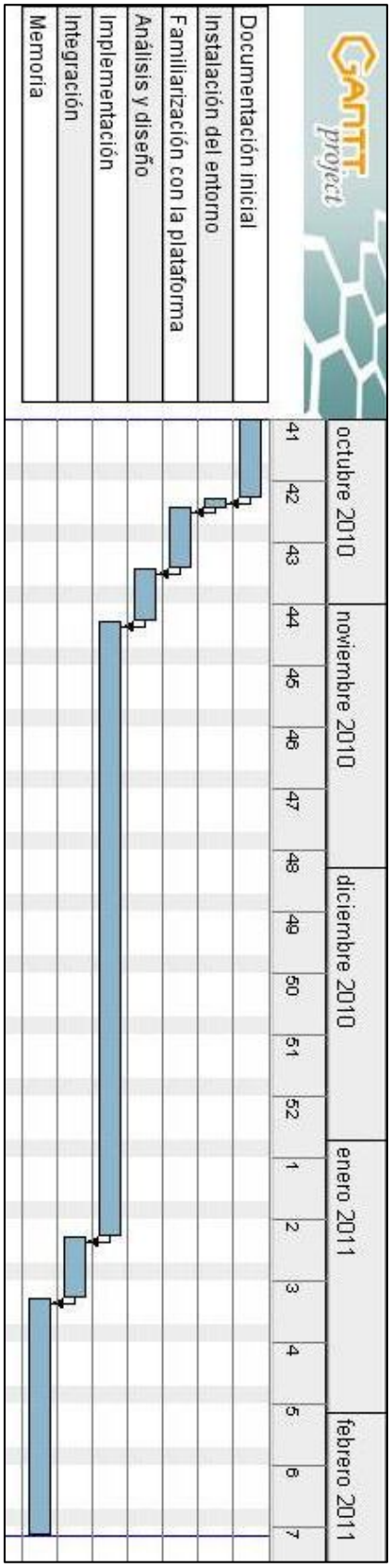


Figura 88. Diagrama de Gantt de la planificación inicial del videojuego

A.2 Planificación inicial individual

Se procede a actuar del mismo modo que en el apartado anterior, individualizando la organización a este proyecto.

La actividad *Implementación* anterior, abarca las siguientes actividades con respecto al módulo desarrollado en este proyecto. De igual modo, se presenta seguidamente una tabla con cada una de esas actividades además de las horas que conllevaría su realización.

HITOS	HORAS
Menú principal	16
Animaciones	16
Efectos de sonido	8
Vibración	4
Créditos	24
Opciones	32
Gestión de los estados	8
Eventos pantalla	16
Protagonista	48
Enemigos	48
Meteoritos	16
Sensor	16
Cámara	24
Escenarios	48
Colisiones	32
Guardar/Cargar configuración	16

Tabla 51: Planificación inicial del proyecto desarrollado

A.2.1 Diagrama de Gantt

Y seguidamente se presenta el diagrama de *Gantt* correspondiente a la planificación inicial individual.

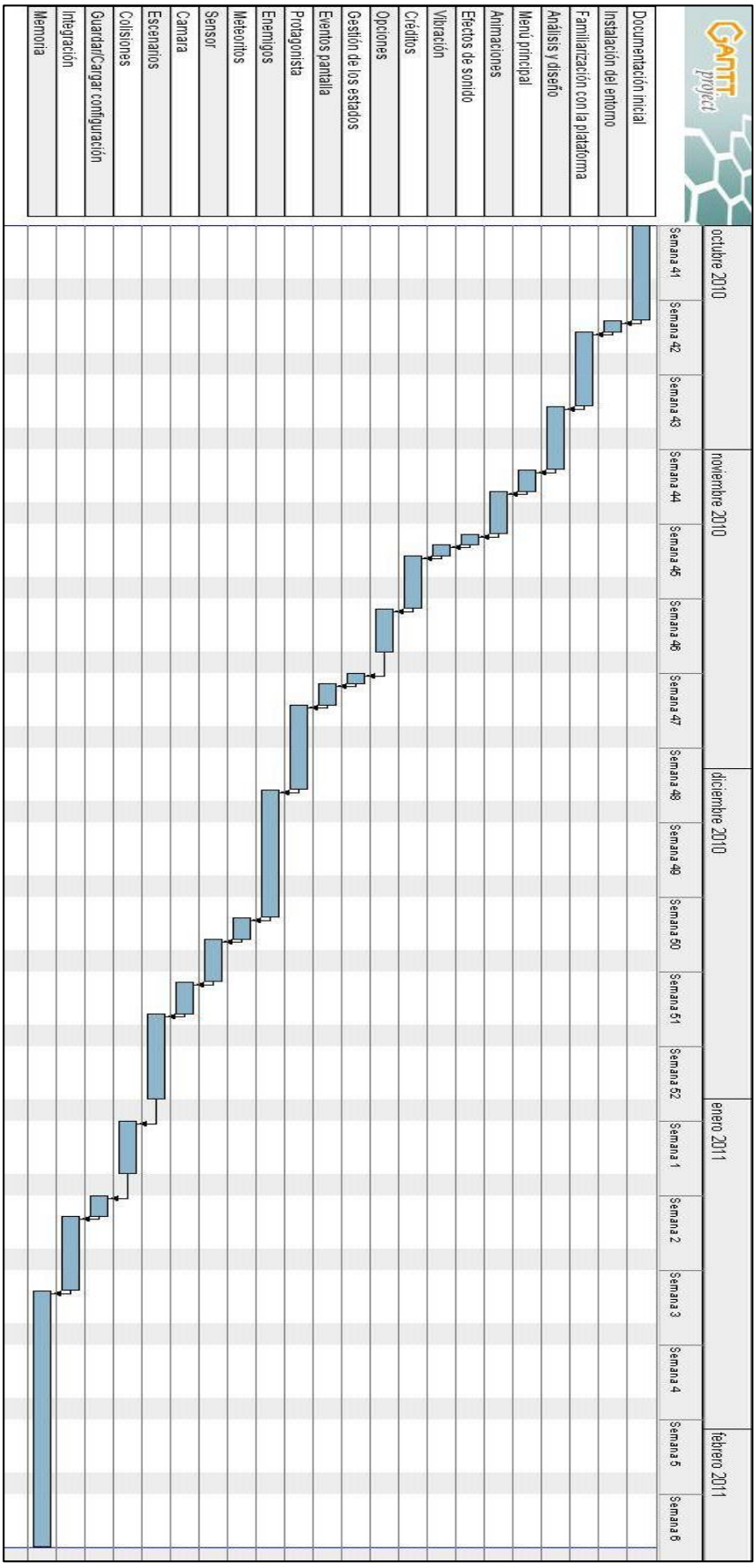


Figura 89. Diagrama de Gantt de la planificación inicial individual

A.3 Planificación final

La planificación inicial que se ha expuesto anteriormente del presente proyecto no se consiguió cumplir. En la Figura 90 se puede observar en cuánto se retrasó el proyecto, reflejando el estado final en el que quedó la planificación.

Las causas que condujeron a este retraso en la planificación inicial que se planteó, fueron varias. La principal fue porque no se tuvo en cuenta que durante el desarrollo del proyecto yo me encontraba matriculado en un *Curso de Adaptación a Grado* para la obtención de otro título universitario, cuya duración se extendía durante todo un año académico. Esto produjo una gran reducción de mí tiempo para poder dedicarme a ello debido a los continuos trabajos, laboratorios y pruebas que se deben realizar en él. Consecuentemente, el tiempo que podía ofrecer era mínimo y no de forma continuada comparado con el horario de 8 horas diarias que se estableció en la planificación inicial y que por lo tanto, no pude dedicarme a ello íntegramente hasta la finalización de dicho curso.

Otra causa, aunque de algo menos trascendencia sobre la planificación, ha sido por no contar con las distintas fiestas que existen en el calendario académico y/o nacional, que aprovechaba para estudiar las múltiples materias que estaba cursando y en cierta manera descansar.

Uno de los hitos en los que se ha producido mayor retraso, comparado con la planificación inicial, ha sido el de *Análisis y Diseño*. El concepto principal que era el de desarrollar un videojuego sobre el Sistema Operativo *Android*, era el único aspecto claro. No se lograba concebir una idea sobre la que se basara el videojuego lo suficientemente original, innovadora y que englobara determinadas características que no se hubieran desarrollado con anterioridad, para llevarse a cabo en un PFC. Se prolongó en más semanas de lo que se había planteado inicialmente, hasta que se llegó a lo que ha sido la idea central del presente proyecto, que es el del desarrollo de la lógica de un videojuego de plataformas sobre escenarios 3D, utilizando para ello la biblioteca desarrollada en otro PFC como interfaz gráfica.

Otro hito que se prolongó en el tiempo ha sido el de *Implementación*. La formación universitaria que he obtenido no está relacionada en su totalidad con el mundo informático, si no con el de la imagen y el sonido. Esto implicaba que las distintas técnicas que se emplean para el desarrollo de videojuegos las desconocía en parte, además de los numerosos aspectos que comprenden. Fue gracias a la ayuda del tutor que se implicó en mi aprendizaje de esas técnicas además de querer que se puliera lo más posible el código resultante para que fuera lo más robusto y eficaz posible. Todo esto provocó que se produjera un retraso en este hito.

Posteriormente, ese código obtenido y pulido se tenía que adaptar a la biblioteca desarrollada por otro PFC que utiliza mi proyecto como interfaz de la lógica desarrollada. Se realizaron algunos cambios del código, también pasos atrás por lo que para la finalización del hito *Integración* se utilizó más tiempo de lo que se planteó inicialmente

HITOS	Horas Iniciales	Horas Finales
Documentación inicial	48	48
Instalación del entorno	6	6
Familiarización con la plataforma	40	40
Análisis y diseño	24	40
Implementación:		
Menú principal	16	32
Animaciones	16	16
Efectos de sonido	8	8
Vibración	4	4
Créditos	24	24
Opciones	32	32
Gestión de los estados	8	12
Eventos pantalla	16	24
Protagonista	48	48
Enemigos	48	48
Meteoritos	16	16
Sensor	16	16
Cámara	24	24
Escenarios	48	48
Colisiones	32	40
Guardar/Cargar configuración	16	24
Integración	40	50
Memoria	150	200

Tabla 52: Comparación entre planificación inicial y final

A.3.1 Diagrama de Gantt

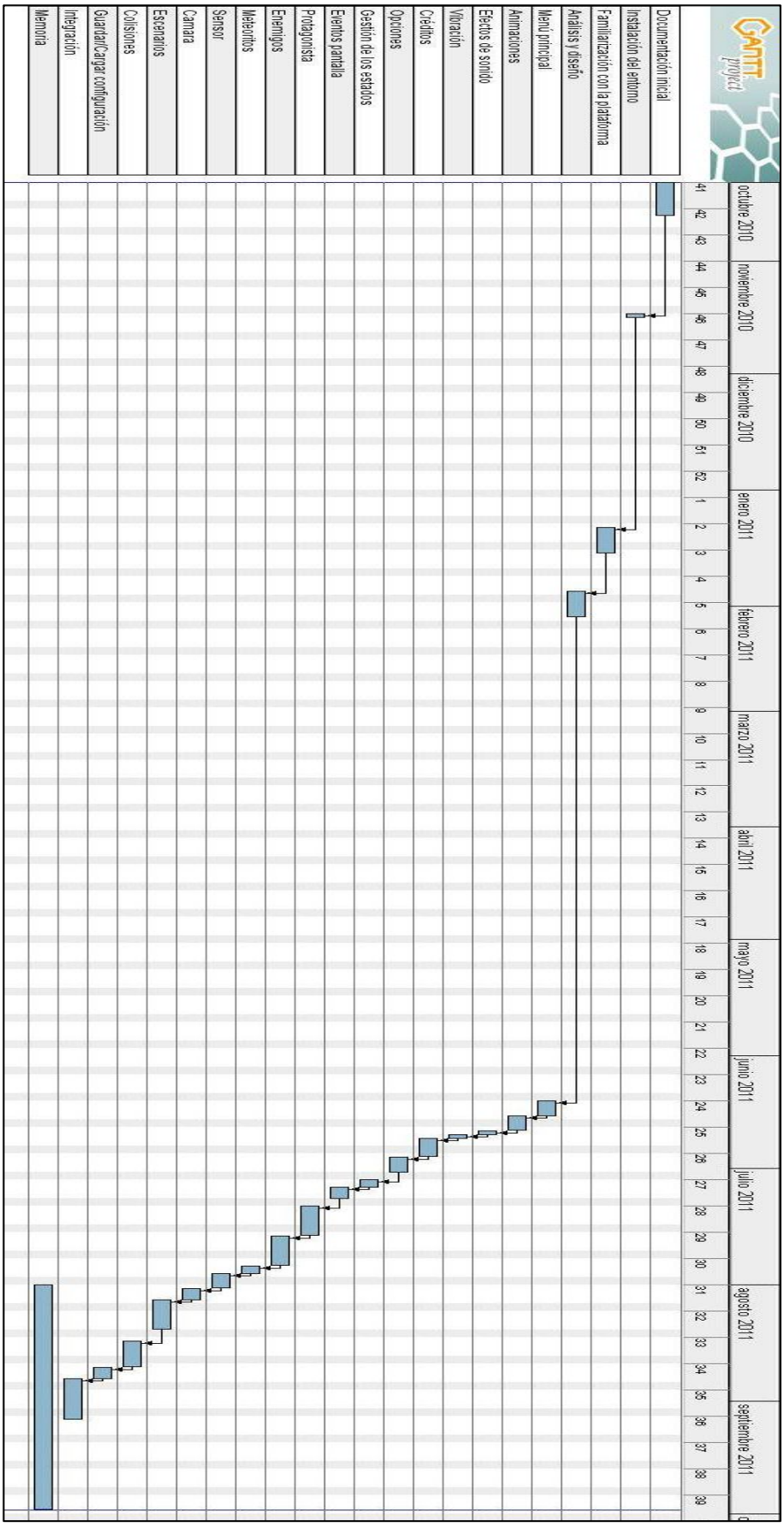


Figura 90. Diagrama de Gantt de la planificación final

Anexo B

Presupuesto

En este anexo se detallan los costes que conlleva la realización del presente proyecto. Los costes se dividen en dos bloques primordialmente, los referidos a los materiales que son el valor total de los componentes empleados en la realización de éste, y los costes de personal, detallándose los honorarios de las personas que han participado, así como del tiempo que han invertido en el trabajo. Y por último se incluye la suma de todos esos costes para proporcionar el coste total del proyecto.

B.1 Costes del videojuego

B.1.1 Costes materiales

En este apartado se detalla el coste de los materiales empleados durante la realización del proyecto. En función de la dedicación de cada material a éste y del coste de su amortización, se calcula su coste imputable sobre el proyecto.

CONCEPTO	PRECIO	DEDICACION	PERIODO DE AMORTIZACION	COSTE AMORTIZADO
Ordenador portátil Samsung, con procesador de doble núcleo y 2GB de memoria RAM	590,00 €	6 Meses	32 Meses	110,62 €
Móvil modelo ZTE Blade	120,00 €	6 Meses	6 Meses	120,00 €
TOTAL	679,00 €			230,62 €

Tabla 53: Coste materiales del videojuego

B.1.2 Costes del personal

Se recoge en la siguiente tabla los costes asociados a los honorarios de las personas que han participado en la realización del proyecto. El trabajo ha sido realizado por el ingeniero proyectista adaptando los roles de *programador* y de *analista* en función de los casos que haya desarrollado, y ha sido dirigido por el tutor del proyecto ejerciendo de *Jefe de Proyecto*.

CARGO	COSTE POR HORA
Programador	15,00 €
Analista	25,00 €
Jefe de Proyecto	35,00 €

Tabla 54: Salarios según especialidad

Si se aplican estos costes al número de horas obtenidas en la planificación inicial del estudio del anexo anterior dedicado por cada especialista en función de las fases de diseño y las tareas encomendadas, el resultado que se obtiene como coste del personal total se muestra en la siguiente tabla:

CARGO	COSTE POR HORA	HORAS	IMPORTE
Programador	15,00 €	412	6.180,00 €
Analista	25,00 €	268	6.700,00 €
Jefe de Proyecto	35,00 €	14	490.00 €
TOTAL		694 horas	13.370,00 €

Tabla 55: Costes personal del videojuego

B.1.3 Coste total

En la siguiente tabla se desglosa el balance final del coste del proyecto:

CONCEPTO	PRECIO
Costes materiales	230,62 €
Costes de la biblioteca de la interfaz gráfica:	
Costes materiales	149,08 €
Costes de personal	16.250,00 €
Costes de personal	13.370,00 €
Subtotal	29.999,7 €
I.V.A. (18%)	5.399,94 €
TOTAL	35.399,64 €

Tabla 56: Costes totales del videojuego

El presupuesto total del videojuego asciende a la cantidad de **TREINTA Y CINCO MIL TRESCIENTOS NOVENTA Y NUEVE** euros con **SESENTA Y CUATRO** céntimos.

B.2 Comparación de costes

En el apartado anterior se ha obtenido el coste total para el desarrollo del videojuego en base a la planificación inicial que se estableció, debido a que ésta es la más cercana a lo que se hubiera empleado en el mundo profesional. Seguidamente se realiza el coste total del videojuego en base a la planificación final obtenida, para de este modo poder realizar su comparación. Se procede de la misma manera que con la planificación inicial, obteniendo los datos de las horas realizadas por cada especialista en la planificación final que se consiguió y de este modo poder realizar los cálculos

CARGO	COSTE POR HORA	HORAS	IMPORTE
Programador	15,00€	466	6.990,00 €
Analista	25,00€	334	8.350,00 €
Jefe de Proyecto	35,00€	14	490,00 €
TOTAL		814 horas	15.830,00 €

Tabla 57: Costes finales personal del videojuego

CONCEPTO	PRECIO
Costes materiales	230,62 €
Costes de la biblioteca de la interfaz gráfica:	
Costes materiales	318.88 €
Costes de personal	24.340,00 €
Costes de personal	15.830,00 €
Subtotal	40.719,50 €
I.V.A. (18%)	7.329,51 €
TOTAL	48.049,01 €

Tabla 58: Coste final total del videojuego

El coste final del videojuego asciende a **CUARENTA Y OCHO MIL CUARENTA Y NUEVE** euros con **UN** céntimo.

TIPO COSTE TOTAL	IMPORTE
Coste Inicial	35.399,64 €
Coste Final	48.049,01 €
DIFERENCIA	12.649,37 €

Tabla 59: Comparación costes totales

Se puede observar que la diferencia obtenida entre los distintos tipos de costes totales del videojuego es amplia en función de que planificación se escoja.

B.3 Publicación del videojuego

El objetivo posterior al desarrollo de un videojuego es el de su publicación para la obtención de beneficios. En este caso solo se puede optar por la única plataforma existente donde poder distribuir la aplicación en *Android*, que es *Android Market*.

Para publicar una aplicación en *Android Market*, al contrario que en *AppStore*, permiten máxima libertad dado que no pasan éstas por estrictas revisiones de funcionamiento, si no que es el propio desarrollador el que se tiene que preocupar que la aplicación funcione en los distintos dispositivos. La publicación es casi instantánea y se debe de realizar un único pago de 25$ para tener licencia de desarrollador para siempre. También se ha de rellenar un formulario dividido en tres partes:

1. La parte de ***assets***, donde se sube el *.apk*, las imágenes promocionales, un gráfico promocional y un video de *Youtube*[109].

2. La parte de ***listing details*** con la descripción de la aplicación en los diferentes idiomas.

3. La parte de ***publishing options*** con la protección y clasificación por edades, además de la información de contacto y aceptaciones correspondientes.

Posteriormente a la realización de todos estos pasos, la aplicación se encontrará publicada en *Android Market*.

El precio que se establece para la descarga de la aplicación es de 1,43€, suponiendo que los porcentajes de ganancia de *Android Market* sobre la aplicación sean del 30%, quedando un 70% para el desarrollador o empresa, se obtiene un beneficio por cada descarga de 1,00€.

Por tanto, si el coste necesario para desarrollar el videojuego es de 35.399,64 €, y se consigue un ingreso de 1,00€ por cada descarga, el número total de ellas necesarias para cubrir los gastos son 35.400. Si se centra este cálculo en la recuperación de los gastos del proyecto que ascienden a 15.340,73 €, se deberán realizar 15.341 descargas de la aplicación.

Cabe recalcar que las ventas de un producto en el mercado no son constantes debido a su evolución durante el tiempo que permanecen en él, pasando por distintas etapas que conforman el *Ciclo de vida del producto*[110].

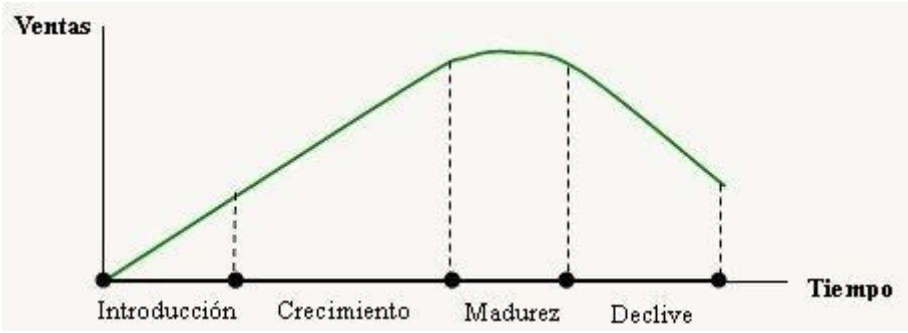


Figura 91. Gráfica etapas del ciclo de vida del producto en el mercado

Las etapas son:

1. **Introducción:** es el momento de lanzamiento del producto, por lo que suele ser una etapa de crecimiento lento debido al estar cargada de incertidumbre y riesgos, ya que el producto aún es poco conocido.

2. **Crecimiento:** una vez introducido el producto y comienza a ser conocido, en esta etapa se produce un considerable aumento en las ventas.

3. **Madurez:** el producto se establece y las ventas siguen siendo altas pero la demanda apenas crece.

4. **Declive:** última etapa de la vida del producto en el que empieza a perder atractivo para los consumidores, disminuyendo las ventas.

Estas etapas se pueden prolongar en el tiempo dependiendo de cómo respondan los usuarios ante la aplicación, pudiendo costar más o menos tiempo en que se conozca el producto, que guste a los usuarios, que se establezca en el mercado… Como se ha analizado en otras secciones de esta memoria, es difícil obtener beneficios mediante la publicación de aplicaciones en *Android Market*, pero se supondrá que el videojuego tendrá muy buena aceptación y los usuarios de *Android* estarán dispuestos a pagarla.

A continuación se realiza un estudio, en base a una posible estimación, de la evolución en ventas durante el primer año que obtendría la aplicación desde su publicación en *Android Market*. Se debe mencionar que la tienda de *Google* está dirigida a todos los usuarios que tengan un dispositivo móvil con S.O. *Android*, por lo que podrán acceder a la aplicación usuarios de los distintos continentes, aumentando con ello la cantidad de descargas que se obtengan.

Como se observa en la gráfica inferior, la etapa de *Introducción* supondría el primer mes de puesta en venta, en el que no se obtendrían gran número de descargas y a partir del segundo es donde comenzaría su crecimiento por la expansión de su popularidad.

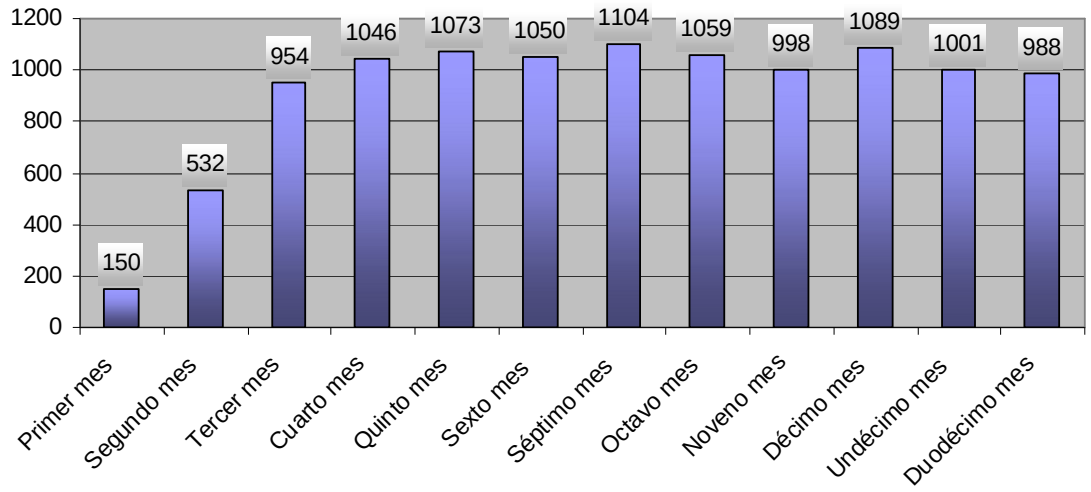


Figura 92. Gráfica ventas primer año en mercado

Las descargas que se lograrían en ese primer año serían de 11.044, por lo que los ingresos que se obtendrían para el desarrollador ascenderían a 11.044 €, cifra algo lejana del coste total del videojuego calculado con anterioridad. Esto supone que la aplicación necesitaría estar en el mercado más tiempo hasta recuperar la inversión inicial, por lo que se estima un tiempo de dos años más a la venta con la misma tendencia de número de ventas.

Anexo C

Manual de usuario

En este anexo se incluye el manual de usuario para que el jugador disponga de la documentación necesaria con el fin de que sepa manejar correctamente el videojuego y sus funcionalidades

C.1 Instalación

Previa instalación de la aplicación, puede obtenerse mediante su descarga en *Android Market* realizando en ella una búsqueda del nombre *Cube´s War*. También puede darse el caso de que el usuario ya disponga del *.apk* del videojuego en la SD Card de su dispositivo móvil. Mediante el uso de alguna aplicación de explorador de carpetas, se selecciona el apk correspondiente y se procede a su instalación. Los pasos a seguir para realizar la instalación del videojuego son los mismos en ambos casos:

1. Se presenta una pantalla preguntando si se quiere instalar la aplicación y los permisos que debe otorgar a ésta para su correcto funcionamiento. Se puede elegir entre dos opciones situadas en la parte inferior de la pantalla, debiendo escoger *Instalar*.

Figura 93. Instalación paso 1

Figura 94. Instalación paso 2

2. Al haber accedido a la instalación de la aplicación, se procede a ésta mientras se muestra por pantalla una barra de ejecución, la cual se visualiza durante la duración del proceso.

3. Una vez acabada la instalación, se confirma la realización de ésta de forma correcta y se ofrece al usuario la posibilidad de ejecutarla en ese momento mediante la opción *Abrir*, o salir del instalador con la opción *Listo*.

Figura 95. Instalación paso 3

C.2 Iniciar la aplicación

En el último paso de la fase de instalación, se ofrece la posibilidad de ejecutar la aplicación en ese mismo momento. Si no se elige esa opción, la aplicación está instalada de igual modo en el dispositivo móvil, por lo que solo hay que buscar su icono dentro de la lista de Apps que tiene el terminal, y seleccionarlo.

Figura 96. Icono de la aplicación *Cube's War*

Una vez seleccionada, se inicia la aplicación. Se muestra una animación de los componentes del menú principal durante una duración determinada, y se visualiza la configuración de éste. Dispone de cuatro opciones con distintas funcionalidades.

MENÚ PRINCIPAL

- **Jugar:** permite iniciar una partida.
- **Opciones:** accede al menú de opciones.
- **Créditos:** muestra a los autores del videojuego desarrollado.
- **Salir:** finaliza la ejecución y cierra la aplicación.

Figura 97. Pantalla menú principal

OPCIONES: El menú de Opciones permite cambiar el estado de varios parámetros que afectan al juego, estableciéndose si se pulsa sobre el botón *Aceptar*.

- **Activar sonidos:** habilita la reproducción de efectos de sonido en la aplicación.
- **Desactivar sonidos:** inhabilita la reproducción de efectos de sonido.
- **Vibración ON/OFF:** in/habilita la vibración del dispositivo móvil ante distintas acciones.

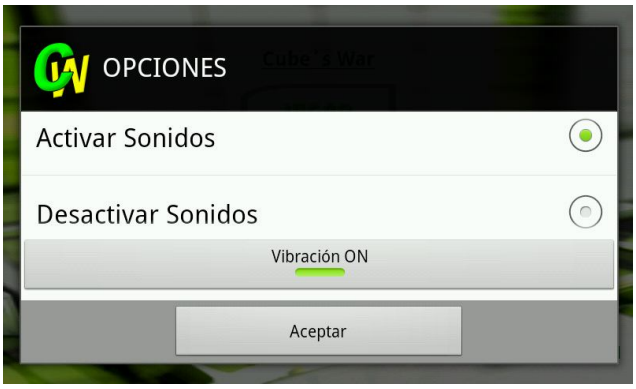


Figura 98. Menú: Opciones

CRÉDITOS: Muestra a los creadores del videojuego. Se debe pulsar sobre la pantalla para volver al menú principal, o pulsar el botón físico ⮌.

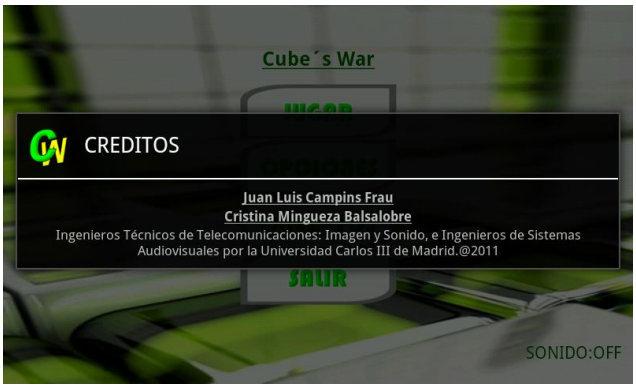


Figura 99. Menú: Créditos

SALIR: muestra una ventana para confirmar la acción que se quiere realizar. Se puede cerrar ésta y volver al menú principal pulsando el botón físico ⮌:

- **Salir:** cierra la aplicación.
- **Cancelar:** cierra la ventana y vuelve al menú principal.

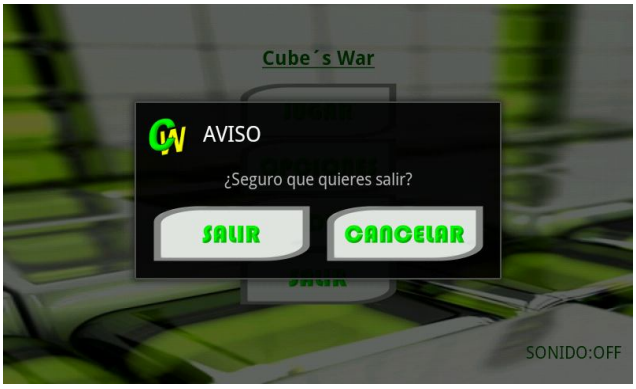


Figura 100. Menú: Salir

JUGAR: inicia una partida. Se muestra la pantalla de videojuego, resaltando la información importante que se muestra durante la partida.

Figura 101. Captura de partida

C.3 Objetivos

El usuario debe encaminar al personaje controlado a lo largo del escenario, mediante el uso de los botones que conforman el pad y el de *Saltar*, para llegar al final de éste donde se ubica una puerta. Si al llegar a ésta no se ha recogido con anterioridad la llave que la abre, no puede pasar al siguiente nivel.

Figura 102. Imagen de las llaves

Figura 103. Imagen de la puerta

En total existen tres niveles que se deben superar, ambientados en distintos escenarios. Si durante el transcurso de la partida el personaje que controla el usuario cae al vacío debido a que se ha acabado la plataforma que se encuentra bajo sus pies, o los enemigos existentes en el escenario le tocan, choca contra los peligrosos meteoritos o se precipita sobre los pinchos, la partida se dará por finalizada y se mostrará la pantalla de *Fin de la partida*.

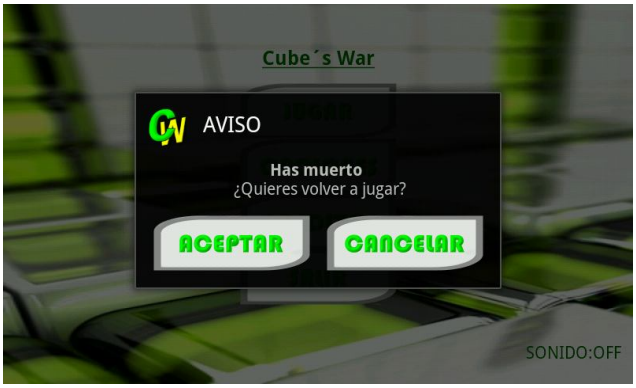


Figura 104. Fin de la partida

Si se elige la opción de *Aceptar*, se reanudará en el escenario donde se encontraba pero desde su inicio. En el caso de que antes de ser eliminado se hubiera recogido la llave

necesaria para pasar de escenario, no hará falta cogerla de nuevo. Por el contrario, si se escoge *Cancelar* o se pulsa el botón físico ⮌, se dirigirá al menú principal.

Si el jugador llega a la puerta del final del escenario habiendo recogido la llave dispuesta en él, superará la fase y se mostrará la pantalla de *Nivel Superado*.

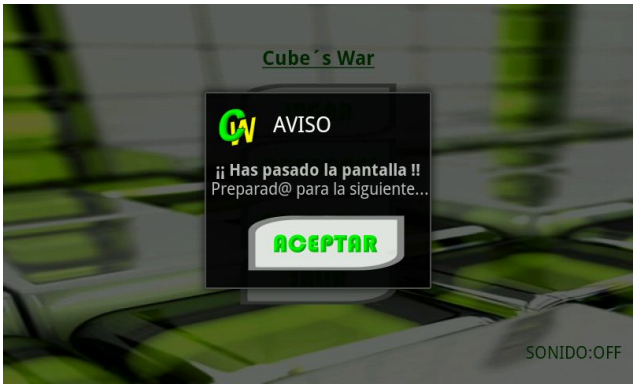


Figura 105. Nivel Superado

Si el escenario superado no es el final, se muestra la ventana con el texto que se visualiza en la imagen superior, cuyo botón *Aceptar* le conducirá al siguiente nivel. Mientras si el escenario es el último, el texto cambia a uno de felicitación, y el botón *Aceptar* le encamina hacia el menú principal.

C.4 Enemigos

A lo largo de las fases pueden aparecer distintos tipos de enemigos. Cada uno de ellos tiene asociada una imagen y un comportamiento. En qué ejes se desplazan y la velocidad en ellos, es lo que define las características del enemigo, enumerándose en la siguiente tabla:

	TIPO 0	TIPO 1	TIPO 2
IMAGEN			
DESPLAZAMIENTO			
Ejes	X	X	X / Z
Velocidad	0.03	0.04	0.03/1.00

Tabla 60: Comparativa de los tipos de enemigos

C.5 Elementos

Además de los enemigos citados, el usuario también puede encontrarse con elementos que debe esquivar, como son los pinchos dispuestos en el escenario sobre los que no se debe dejar caer, y los meteoritos que descienden de la parte superior de éste.

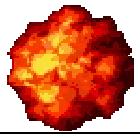
TIPO	PINCHO	METEORITO
IMAGEN		
DESPLAZAMIENTO		
Ejes	Ninguno	Y
Velocidad	Estático	0.2

Tabla 61: Comparativa de los elementos